



Current Topics in Computer Science

BUSINESS INTELLIGENCE PROJECT REGARDING SALES

Instructor: Dr. Phan Trong Nhan

<i>Student 1:</i>	Nguyen Dai Minh	10421037
<i>Student 2:</i>	Nguyen Khoi Nguyen	10421097
<i>Student 3:</i>	Nguyen Ho Tan Dat	10421071
<i>Student 4:</i>	Nguyen Truc Linh	10421088
<i>Student 5:</i>	Dao The Hien	10421074

December 19, 2024

Content

1	Introduction	3
2	Investigate the topics of sales	3
2.1	Sales Definition	3
2.2	Popular metrics to measure sales	3
3	Analyze the data source	4
4	Problem Statement	9
5	Data Warehouse design	9
5.1	Date Dimension	11
5.2	Geography Dimension	12
5.3	Customer Dimension	13
5.4	Product Dimension	14
5.5	Sales Territory Dimension	15
5.6	Promotion Dimension table	16
5.7	Sales fact table	17
6	ETL Pipeline Implementation	18
6.1	DimCustomer	18
6.1.1	Control Flow	18
6.1.2	Data Flow	19
6.2	DimDate	20
6.2.1	Control Flow	20
6.2.2	Data Flow	21
6.3	DimProduct	22
6.3.1	Control Flow	22
6.3.2	Data Flow	23
6.4	DimSalesTerritory	24
6.4.1	Control Flow	24
6.4.2	Data Flow	25
6.5	DimPromotion	26
6.5.1	Control Flow	26
6.5.2	Data Flow	27
6.6	FactSales	28
6.6.1	Control Flow	28
6.6.2	Data Flow	29
6.7	Scheduling	30

7	Data Analysis	31
7.1	Data Extraction	31
7.2	Date Preprocessing	32
7.3	Visualization and Key insights	33
7.3.1	Datasets that group by OrderDate and TerritoryID	33
7.3.2	Datasets that group by OrderDate and ProductCategoryName	40
7.4	Hypothesis Testing	44
7.5	Regression Analysis	46
7.5.1	Data Preparation and Preprocessing	46
7.5.2	Model Selection and Hyperparameter Tuning	47
7.5.3	Evaluation Metrics	49
7.5.4	Final Predictions and Model Performance	50
7.5.5	Future Predictions (Daily and Weekly)	51
7.5.6	Load data to data warehouse	54
7.5.7	Notebook scheduling	55
8	Use-case and recommendations	55
8.1	Addressing Low Performance in Germany and France	55
8.2	Addressing Underperformance in Accessories	56
9	Power BI Dashboard	57

1 Introduction

This business intelligence project's goals include:

- Investigate the topics of sales and analyze the data source
- Problem statement
- Filter out necessary features from the raw datalake and design corresponding data warehouse
- Implementing ETL pipeline with incremental loading and near real-time
- Discover insights through data visualizations, hypothesis testing and regression analysis
- Build a BI dashboard using Power BI
- Show one use-case and deliver a recommendation to support business

2 Investigate the topics of sales

2.1 Sales Definition

In business operations, sales refer to any transactions where money or value is exchanged for the ownership of a good or entitlement to a service. In an accounting context, sales refers to a company's revenue earned from the sales of products or services (net sales). In conclusion, to measures sales we can have a lot of methods to measures it.

2.2 Popular metrics to measure sales

Here are some popular metrics that we will be using to report about sales:

- Total Revenue: Total revenue is the total receipts a seller can obtain from selling goods or services to buyers. It can be calculated by using:

$$ProductPrice \times ProductSoldQuantity$$

- Average profit margin: Average Profit Margin is a financial metric used to measure the profitability of a company relative to its sales. It represents the average amount of profit that a company earns from its revenue. In other words, it shows what percentage of sales revenue remains after all expenses are deducted. It can be calculated by using:

$$\frac{TotalProfit}{TotalRevenue} \times 100$$

- Average revenue per user: Average Revenue Per User (ARPU) is a metric used to measure the revenue generated per user or unit in a given time period. It is commonly used in

industries like telecommunications, software, media, and subscription-based services to assess how much revenue each customer or user contributes on average. It can be calculated by using:

$$\frac{TotalRevenue}{TotalCustomer}$$

3 Analyze the data source

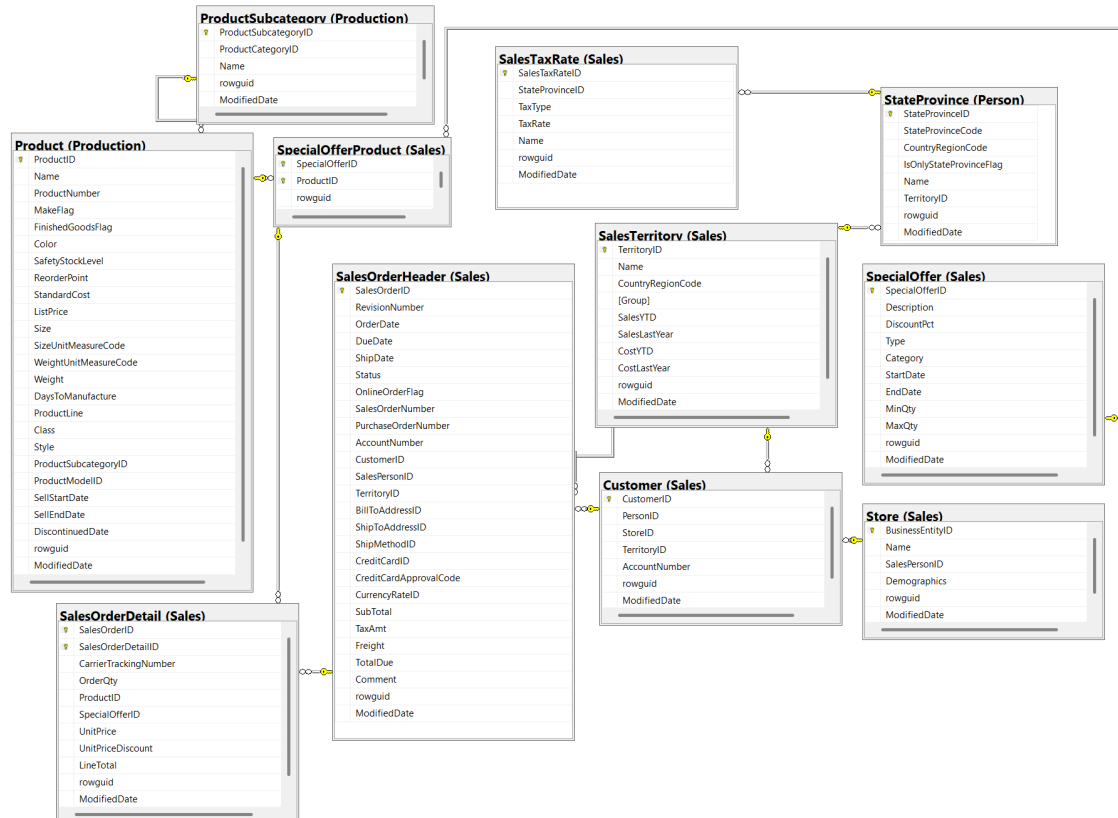


Figure 1. The OLTP diagram of all metrics

This figure show us the relationship of all the OLTP tables that we are going to use. As can be seen, the product table have subcategory and is referenced by special offer product. That table is then referenced in SalesOrderDetail and SalesOrderHeader. SalesOrderHeader table reference 2 other tables which is salesTerritory and Customer. Customer is then reference store and salesTerritory include stateProvince and taxRate.

Based on the 4 sales metrics above, let's start analyzing our data sources for some insights. Firstly, we must look at the span of time recorded in this database. Here is the query and results:

```

SELECT TOP (1)
    [OrderDate]
FROM [CompanyX].[Sales].[SalesOrderHeader]
ORDER BY
    [OrderDate] asc;
    
```

OrderDate
2011-05-31 00:00:00.000

Figure 1. Query for the oldest order date and results

```

SELECT TOP (1)
    [OrderDate]
FROM [CompanyX].[Sales].[SalesOrderHeader]
ORDER BY
    [OrderDate] DESC;

```

OrderDate
2014-06-30 00:00:00.000

Figure 2. Query for the recent order date and results

As can be seen the date recorded in the database only cover a period of approximately 3 years, therefore it would be reasonable for us to aimed at predicting the monthly amount of the aforementioned metrics. Secondly, let's begin exploring the amount of countries and regions available in the dataset. Here is the query and results:

```

-- Count the total number of distinct countries in the table
SELECT COUNT(DISTINCT CountryRegionCode) AS TotalCountries
FROM [CompanyX].[Sales].[SalesTerritory];

-- Count the number of distinct countries in each Group
SELECT [Group], COUNT(DISTINCT CountryRegionCode) AS CountriesCount
FROM [CompanyX].[Sales].[SalesTerritory]
GROUP BY [Group];

```

Name	CountryRegionCode	Group
Northwest	US	North America
Northeast	US	North America
Central	US	North America
Southwest	US	North America
Southeast	US	North America
Canada	CA	North America
France	FR	Europe
Germany	DE	Europe
Australia	AU	Pacific
United Kingdom	GB	Europe

Group	CountriesCount
Europe	3
North America	2
Pacific	1

TotalCountries
6

Figure 3. Queries for the amount of regions and countries the company sells to.

We can see that the company only sell their product to 3 regions: North America, Europe and Pacific and there are only 3 country in Europe which is France, Germany and the UK. In NA, we only have the US and Canada, in the Pacific, we only have Australia. Thirdly, we will explore the data related to sales. Here is the query and results:

```
SELECT COUNT(PersonID)
FROM [CompanyX].[Sales].[Customer]
WHERE [StoreID] IS NULL and [PersonID] is NOT NULL;
```

18484

Figure 4. Query for the amount of customers online.

```
SELECT COUNT(StoreID)
FROM [CompanyX].[Sales].[Customer]
WHERE [StoreID] IS NOT NULL and [PersonID] is NULL;
```

701

Figure 5. Query for the amount of resellers.

In the database, we found that we have a distinction between Internet sales as well as sales to a distributor such as some resellers. We discover that the amount of customers ordering through the internet is 18484 and the amount of stores that are buying products in bulk is 701. For the sake of simplicity, we will not be separating between the two types of sales, and this will reflect in our datawarehouse schema. Finally, let's take a look at some of the metrics related to product. Here is our query and results:

```
SELECT
    COUNT(DISTINCT ProductID) AS UniqueProductCount
FROM
    [CompanyX].[Sales].[SalesOrderDetail];
```

UniqueProductCount
266

Figure 6. Query for the amount of unique products sold by the company.

```
SELECT p.ProductID, p.Name, SUM(s.OrderQty) as Total
FROM Production.Product p
JOIN Sales.SalesOrderDetail s
ON p.ProductID = s.ProductID
GROUP BY p.ProductID, p.Name
ORDER BY Total DESC
```

	ProductID	Name	Total
1	712	AWC Logo Cap	8306
2	870	Water Bottle - 30 oz.	6815
3	711	Sport-100 Helmet, Blue	6740
4	715	Long-Sleeve Logo Jersey, L	6582
5	708	Sport-100 Helmet, Black	6528
6	707	Sport-100 Helmet, Red	6261
7	864	Classic Vest, S	4247
8	873	Patch Kit/8 Patches	3865
9	884	Short-Sleeve Classic Jersey, XL	3864
10	714	Long-Sleeve Logo Jersey, M	3634
11	859	Half-Finger Gloves, M	3464
12	863	Full-Finger Gloves, L	3378
13	877	Bike Wash - Dissolver	3319

Figure 7. Query for the total quantity of products sold.

```
SELECT p.ProductID, p.Name,
SUM((UnitPrice - UnitPriceDiscount) * OrderQty) AS TotalPrice
FROM Production.Product p
JOIN Sales.SalesOrderDetail s
ON p.ProductID = s.ProductID
GROUP BY p.ProductID, p.Name
Order by TotalPrice desc
```

	ProductID	Name	TotalPrice
1	782	Mountain-200 Black, 38	4406146.9062
2	783	Mountain-200 Black, 42	4014064.1199
3	779	Mountain-200 Silver, 38	3696484.3126
4	780	Mountain-200 Silver, 42	3441290.3343
5	781	Mountain-200 Silver, 46	3436089.3546
6	784	Mountain-200 Black, 46	3311097.3185
7	793	Road-250 Black, 44	2518298.6241
8	794	Road-250 Black, 48	2348245.6523
9	795	Road-250 Black, 52	2012447.775
10	753	Road-150 Red, 56	1839230.78
11	976	Road-350-W Yellow, 48	1788321.1865
12	749	Road-150 Red, 62	1769096.688
13	969	Touring-1000 Blue, 60	1728802.7105
14	973	Road-350-W Yellow, 40	1663510.5603
15	966	Touring-1000 Blue, 46	1592387.8051
16	792	Road-250 Red, 58	1587008.1825
17	957	Touring-1000 Yellow, 60	1578597.4019
18	751	Road-150 Red, 48	1540803.062
19	796	Road-250 Black, 58	1506377.6325

Figure 8. Query for the sales made by each product.

We have found that there is 266 distinct products sold by the company which is a huge variety

and most of the products are related to biking. We can also see that the most frequently sold in terms of quantity products belong to the accessories category while the most frequently sold products in terms of sales are categorized as bikes.

After looking through all the tables that we need, we need to also check whether the tables themselves are clean and usable. Here is all the queries and what we found:

```
FROM [CompanyX].[Sales].[SalesOrderHeader]
WHERE SalesOrderID IS NULL OR RevisionNumber IS NULL OR OrderDate IS NULL OR DueDate IS NULL OR ShipDate IS NULL OR Status IS NULL OR On
```

CurrencyRateID
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL

Figure 9. Query for missing data in SalesOrderHeader.

```
FROM [CompanyX].[Sales].[SalesOrderDetail]
WHERE CarrierTrackingNumber IS NULL OR OrderQty IS NULL OR ProductID IS NULL OR SpecialOfferID IS NULL OR UnitPrice IS NULL OR UnitPrice
```

CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice
918F-49F3-AD	1	760	1	NULL
918F-49F3-AD	2	770	1	NULL
918F-49F3-AD	2	715	1	NULL
FF1F-4DD0-98	NULL	760	1	419.4589
FF1F-4DD0-98	NULL	756	1	874.794
FF1F-4DD0-98	NULL	738	1	178.5808
FF1F-4DD0-98	NULL	707	1	20.1865
FF1F-4DD0-98	NULL	729	1	183.9382
FF1F-4DD0-98	NULL	711	1	20.1865
FF1F-4DD0-98	NULL	715	1	28.8404
FF1F-4DD0-98	NULL	725	1	183.9382
FF1F-4DD0-98	NULL	758	1	874.794
FF1F-4DD0-98	NULL	761	1	419.4589
FF1F-4DD0-98	NULL	753	1	2146.962
FF1F-4DD0-98	NULL	708	1	20.1865
FF1F-4DD0-98	NULL	765	1	419.4589
FF1F-4DD0-98	NULL	712	1	5.1865
FF1F-4DD0-98	NULL	763	1	419.4589
NULL	1	749	1	3578.27
NULL	1	773	1	3399.99
NULL	1	773	1	3399.99

Figure 10. Query for missing data in SalesOrderDetails.

From the queries above, it is discovered that in the table salesOrderHeader have some null values in CurrencyRateID. Even though without currency rate it is not possible to convert to different currency but the database itself use USD as the default currency so it should not be too important. In SalesOrderDetail there are quite a few missing which is CarrierTrackingNumber, OrderQty and UnitPrice. OrderQty and UnitPrice might be useful for calculate sales but we already have LineTotal so it might not be necessary. When it comes to CarrierTrackingNumber, as we only cares about sales, it does not really matter with NULL. But we can guess that those that buy from the internet will have CarrierTrackingNumber

4 Problem Statement

After some surface investigations, our project will be aimed at designing a data warehouse based on the 3 aforementioned metrics, implementing an ETL pipeline to ingest data from the database CompanyX to the newly designed data warehouse, conducting EDA using Python including univariate analysis, multivariable analysis, hypothesis testing and regression analysis to find, validate insights as well as forecast the 3 aforementioned metrics specific products. Finally, a Power BI dashboard will be shown to showcase and highlight the insights as well as the regression model.

5 Data Warehouse design

Here is our overall diagram for the data warehouse schema:

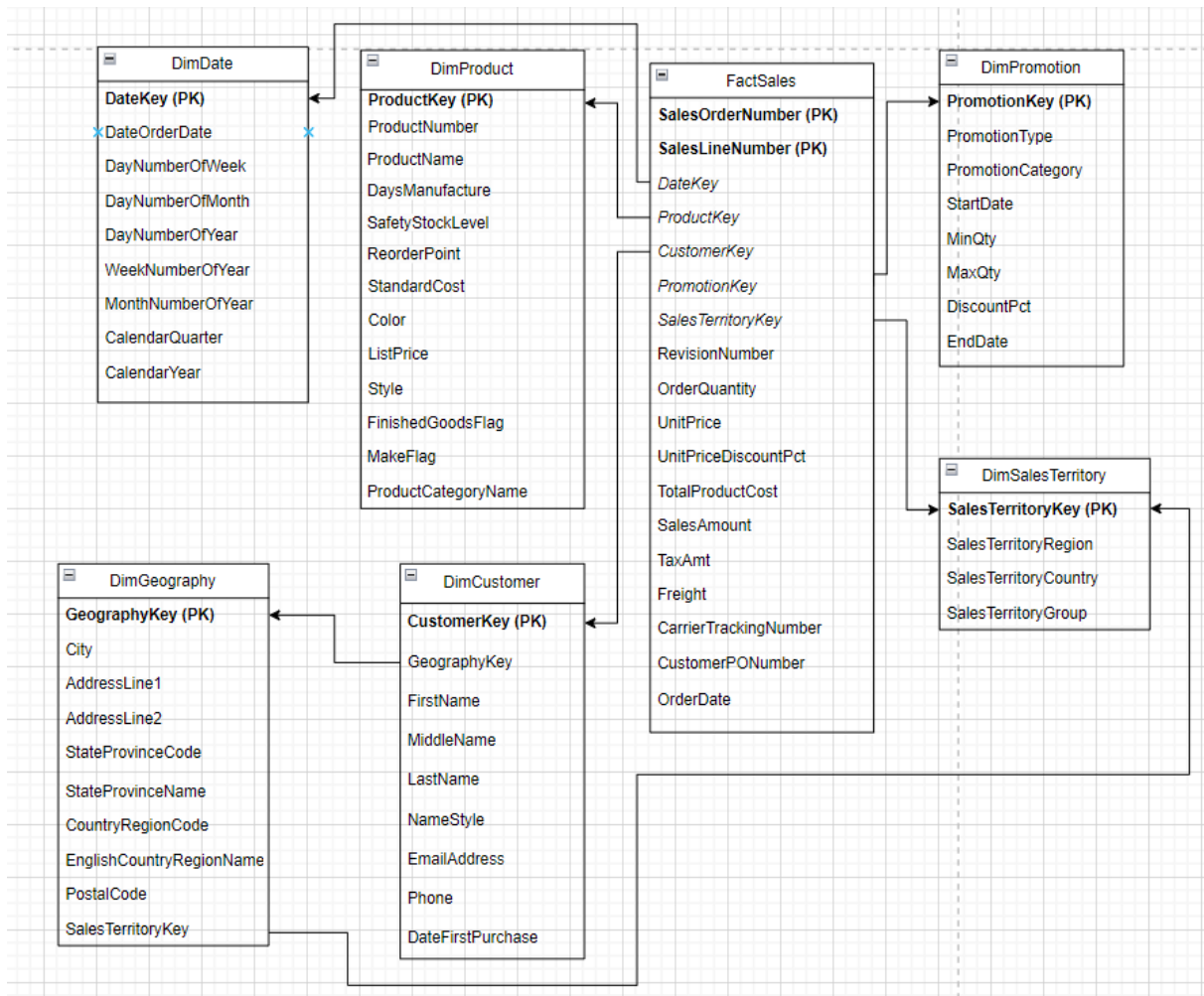


Figure 1: Star Schema

We have 1 main sales fact table with 5 dimensions. Four of the dimension tables which includes date, customer, product and sales territory is connected to the sales fact table while the geography dimension is connected to the customer dimension and sales territory dimensions.

5.1 Date Dimension

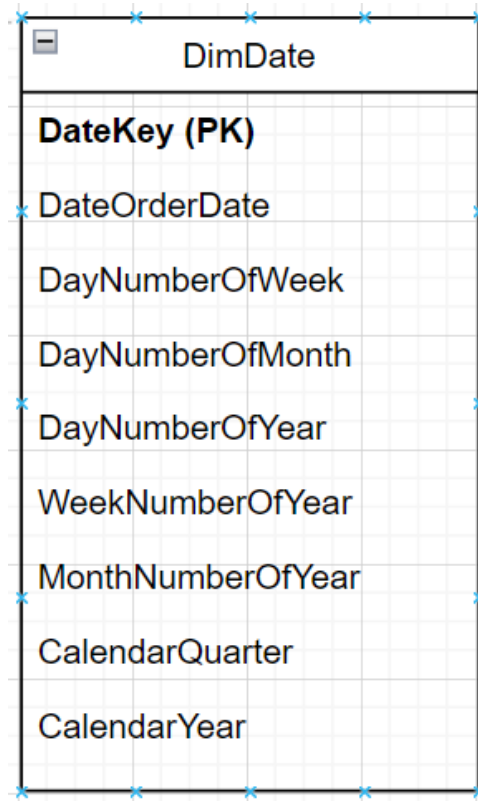


Figure 2: Date Dimension

The DimDate table has 1 primary key to uniquely identify each date. Here is a brief description of each columns:

- DateKey is a primary key number columns for each date of the periods of sales record from the sales order detail tables
- DateOrderDate is the original OrderDate column showing the order date
- DayNumberOfWeek indicate which date it is in terms of week
- DayNumberOfMonth indicate which date it is in terms of month
- DayNumberOfYear indicate which date it is in terms of year
- WeekNumberOfYear indicate which week it is in terms of year
- MonthNumberOfYear indicate which month it is in terms of year
- CalendarQuarter indicate which quarter of year it is
- CalendarYear indicate the year of the date

5.2 Geography Dimension

DimGeography
GeographyKey (PK)
City
AddressLine1
AddressLine2
StateProvinceCode
StateProvinceName
CountryRegionCode
EnglishCountryRegionName
PostalCode
SalesTerritoryKey

Figure 3: Geography Dimension

The DimGeography table has 1 primary key GeographyKey. Here is a brief description of each columns:

- GeographyKey is a auto increment always generated columns to uniquely identify each records of location
- City indicate the city
- StateProvinceCode indicate the code number of the state/province
- StateProvinceName indicate the name of the state/province
- AddressLine1 indicate the first address
- AddressLine2 indicate the second address
- CountryRegionCode indicate the code number of the region
- EnglishCountryRegionName indicate the name of the region
- PostalCode indicate the postal code
- SalesTerritoryKey is a foreign key referencing the sales territory dimension table

5.3 Customer Dimension

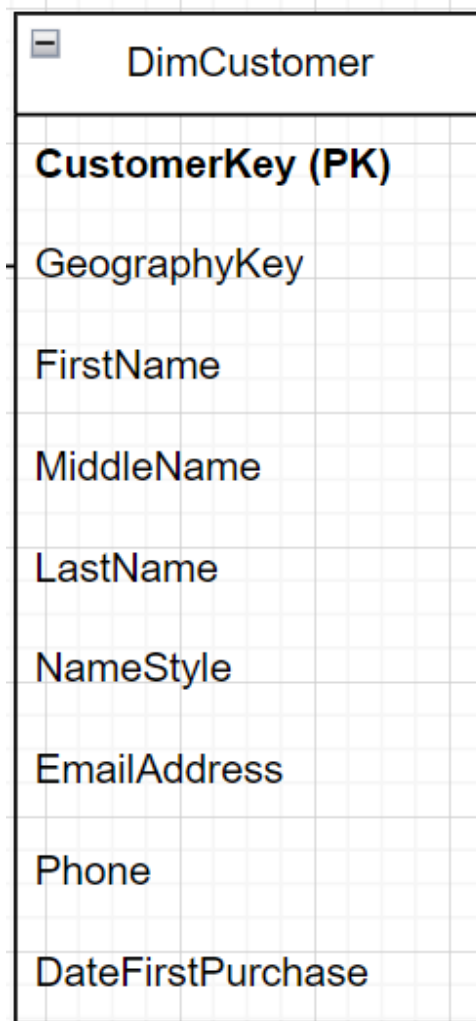
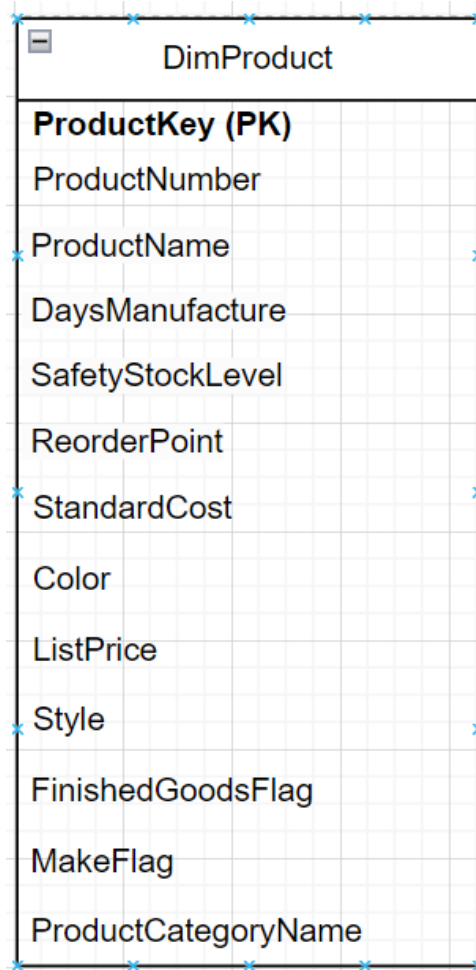


Figure 4: Customer Dimension

The DimCustomer table also has 1 primary key to identify each different customer. Here is a brief description of each columns:

- CustomerKey indicate the id of the customer
- GeographyKey indicate their location and is a foreign key connected to the DimGeography
- FirstName contains the first name of customer
- MiddleName contains the middle name of customer
- LastName contains the last name of customer
- NameStyle indicate the name style
- EmailAddress contains the email address of customer
- YearlyIncome contains the yearly income of customer
- DateFirstPurchase contain the first ever purchase date of the customer

5.4 Product Dimension



ProductKey (PK)
ProductNumber
ProductName
DaysManufacture
SafetyStockLevel
ReorderPoint
StandardCost
Color
ListPrice
Style
FinishedGoodsFlag
MakeFlag
ProductCategoryName

Figure 5: Product Dimension

The DimProduct table has 1 primary key ProductKey. Here is a brief description of each column:

- ProductKey is an unique identifier of each product
- ProductNumber contains a code number of a product
- ProductName indicates the product's name
- DaysManufacture contains the amount of days it takes to produce a product
- SafetyStockLevel indicates the amount of stock that is considered safe
- ReorderPoint is the amount of product indicating a need for restocking
- StandardCost is the cost for production
- Color is the color of product
- ListPrice is the suggested listing price

- Style is the style of product
- FinishedGoodsFlag indicate whether a product is finished
- MakeFlag indicate whether a product is in production
- ProductCategoryName is the category of the product

5.5 Sales Territory Dimension

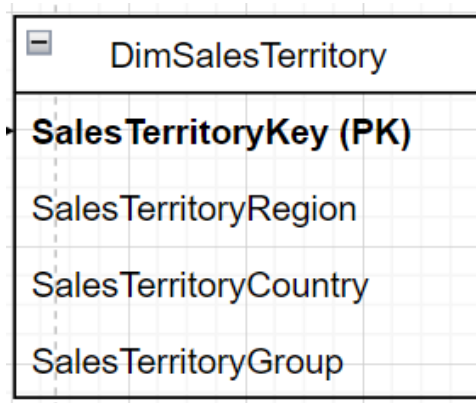


Figure 6: Sales Territory Dimension

The DimSalesTerritory table has 1 primary key SalesTerritory key. Here is a brief description of each column:

- SalesTerritory indicate unique id for each territory
- SalesTerritoryRegion indicate the region
- SalesTerritoryCountry indicate the country
- SalesTerritoryGroup indicate the group

5.6 Promotion Dimension table

DimPromotion
PromotionKey (PK)
PromotionType
PromotionCategory
StartDate
MinQty
MaxQty
DiscountPct
EndDate

Figure 7: Promosion Dimension

The DimPromotion table has 1 primary key. Here is a brief description of each column:

- PromotionKey is the unique identifier of each promotion
- PromotionType indicate the type of promotion
- PromotionCategory is the category of promotion
- StartDate and EndDate is the duration of promotion
- MinQty and MaxQty is the max and min amount that the promotion can be applied
- DiscountPct is the percentage discount

5.7 Sales fact table

FactSales
SalesOrderNumber (PK)
SalesLineNumber (PK)
<i>DateKey</i>
<i>ProductKey</i>
<i>CustomerKey</i>
<i>PromotionKey</i>
<i>SalesTerritoryKey</i>
RevisionNumber
OrderQuantity
UnitPrice
UnitPriceDiscountPct
TotalProductCost
SalesAmount
TaxAmt
Freight
CarrierTrackingNumber
CustomerPONumber
OrderDate

Figure 8: Sales Fact Table

The FactSales has 2 primary key as a composite key SalesOrderNumber and SalesLineNumber. Here is a brief description:

- SalesOrderNumber is the id of the number but because a number can have multiple product so we must use another identifier.
- SalesLineNumber is an identifier created by numbering each of the product of salesOrder-

Number to separate each order to be unique

- DateKey, ProductKey, CustomerKey, PromotionKey, SalesTerritoryKey are all foreign key reference to the dimension table of each respected key
- RevisionNumber is the revision number
- OrderQuantity is number quantity of the product ordered
- UnitPrice is the price of each unit
- UnitPriceDiscountPct is the discount percent of each product
- TotalProductCost is the production cost
- SalesAmount is the sales total
- TaxAmt is tax calculated by taking SalesAmount divide by 12.5 assuming the tax is 8%
- Freight is calculated by taking SalesAmount divide by 40 assuming the tax is 2.5%
- CarrierTrackingNumber is the shipping number
- CustomerPONumber is the purchase order number of customer
- OrderDate is the order date of the sales order

6 ETL Pipeline Implementation

6.1 DimCustomer

6.1.1 Control Flow

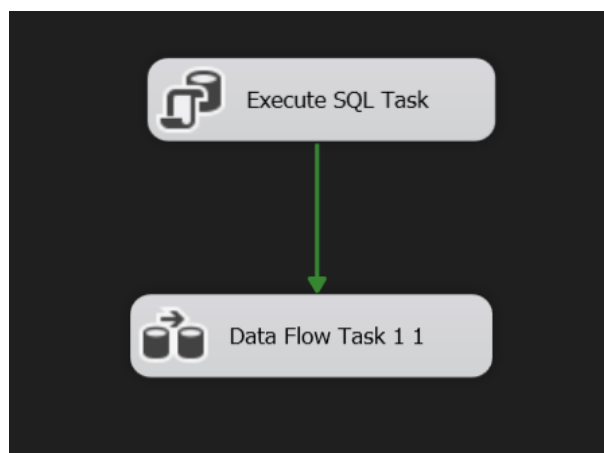


Figure 9: Control Flow for DimCustomer

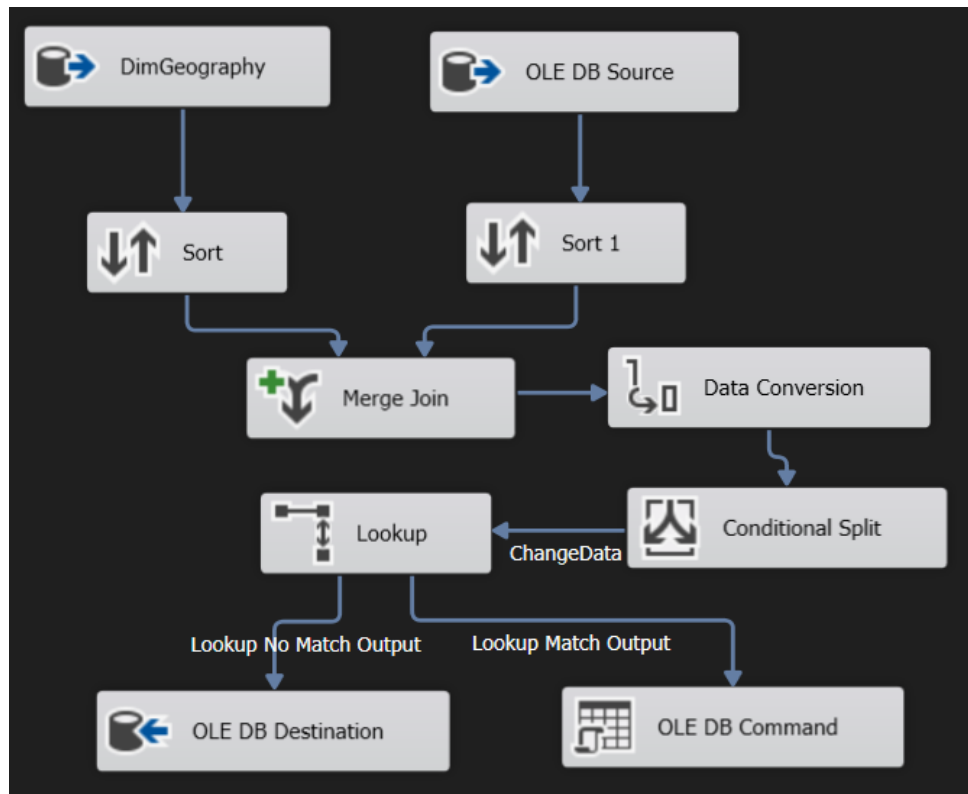


Figure 11: Data Flow for DimCustomer

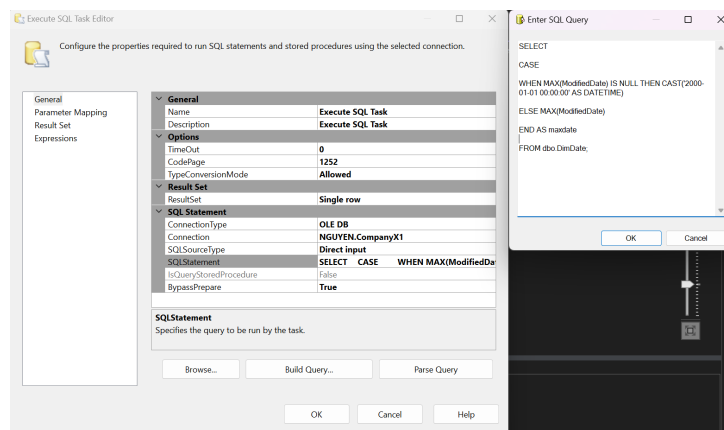


Figure 10: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the DimCustomer and create a system variable VarMaxDate to use as comparison in the data flow

6.1.2 Data Flow

In general, the data flow of DimCustomer are as follow:

1. Select the data source from the DimGeography and multiple tables from CompanyX database
2. Sort the table using the variables used for merging

3. Join the 2 table using those variables
4. Convert data that is not the correct types for incremental loading
5. Use conditional split and compare the modifiedDate columns with the latest modifiedDate of the DimCustomer will help us identify the newest rows
6. Use look up to match the unique identifier to know whether the rows exist or not
7. If the rows already exist update the table using OLE DB Command
8. If the rows don't exist add the new rows to the table

6.2 DimDate

6.2.1 Control Flow

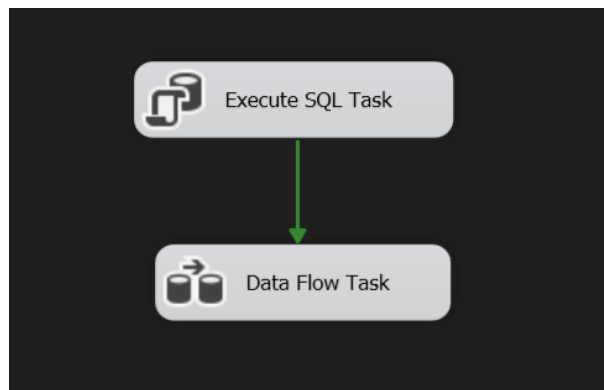


Figure 12: Control Flow for DimDate

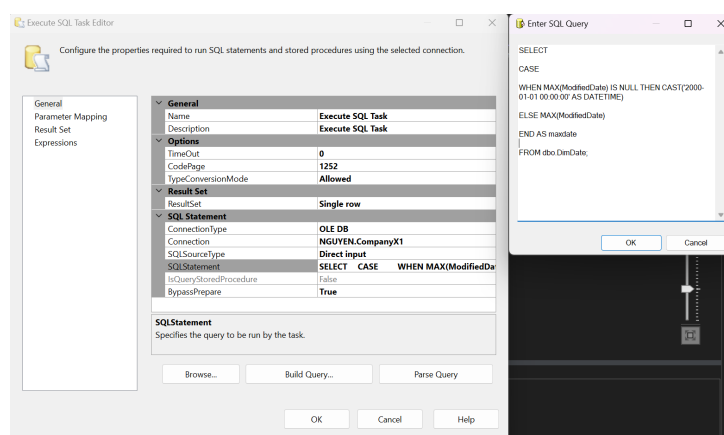


Figure 13: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the DimDate and create a system variable VarMaxDate to use as comparison in the data flow.

6.2.2 Data Flow

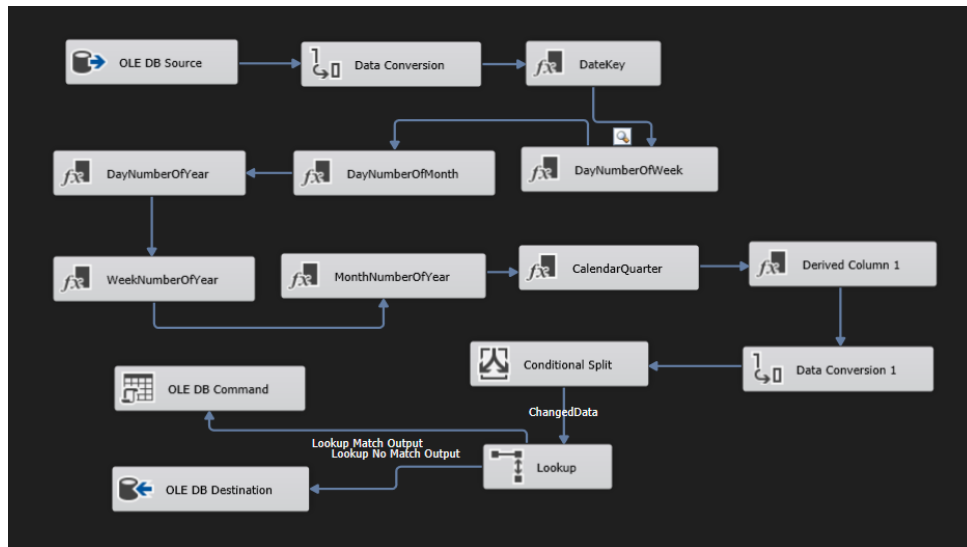


Figure 14: Data Flow for DimDate

In general, the flow of data of DimDate are as follow:

1. Select the column OrderDate, ModifiedDate in the Sales.SalesOrderHeader so we can derive a lot of column for our date dimension.
2. Convert the columns into appropriate format
3. DayNumberOfWeek use datepart(weekday) function to figure out the number of day in a week
4. DayNumberOfMonth use day() function to figure the the number of day in a month
5. DayNumberOfYear use datepart(dayofyear) function the number of day in a year
6. WeekNumberOfYear use datepart(week) function to figure the week number in a year
7. MonthNumberOfYear use datepart(month) function to figure out the month number in a year
8. CalendarQuarter use quarter() function to figure out the quarter of the year
9. CalendarYear use datepart(year) function to figure out the year
10. Use DataConversion to convert ModifiedDate into suitable format
11. Use Conditional Split to compare VarMaxDate and ModifiedDate to identify the new rows
12. Use Lookup to match the ID with the existing rows
13. If the rows already exist then update the rows
14. If the rows dont exist then insert the new rows into the table

6.3 DimProduct

6.3.1 Control Flow

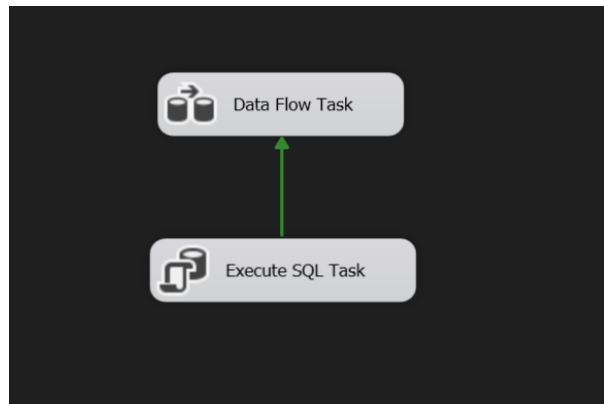


Figure 15: Control Flow for DimProduct

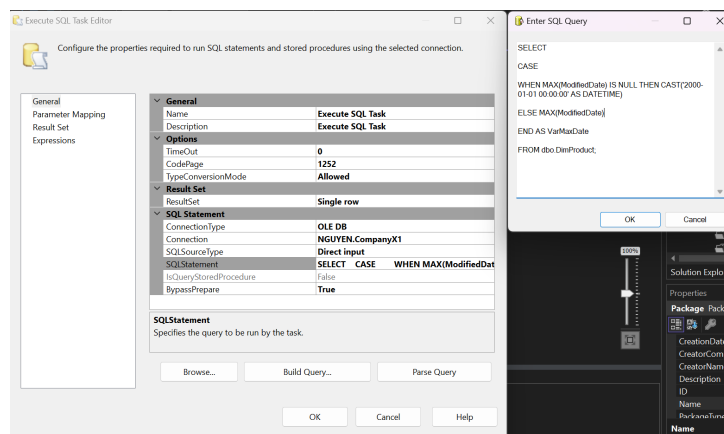


Figure 16: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the DimProduct and create a system variable VarMaxDate to use as comparison in the data flow.

6.3.2 Data Flow

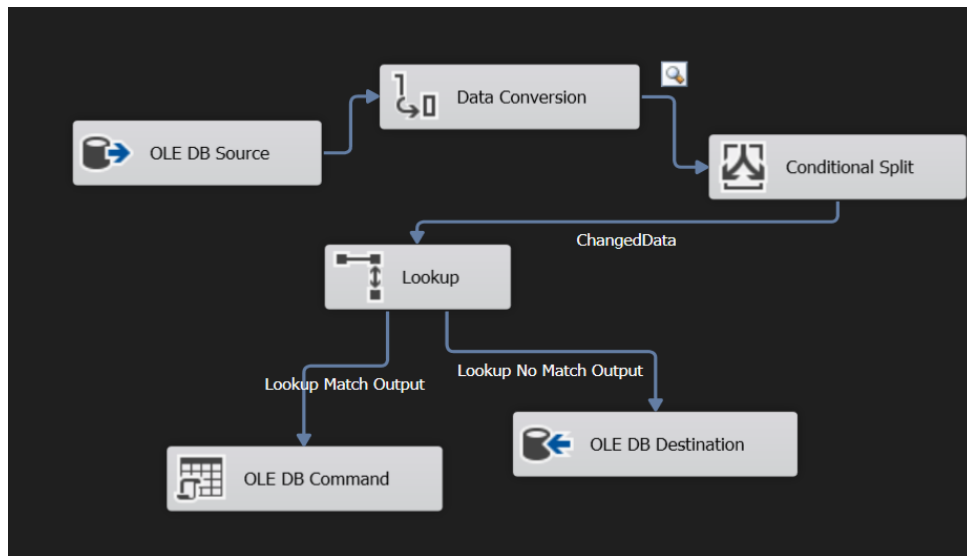


Figure 17: Data Flow for DimProduct

In general, the flow of data of DimProduct are as follow:

1. Select the column in the schema by joining the following tables:
 - Production.Product
 - Production.ProductSubCategory
 - Production.ProductCategory
2. Convert the ModifiedDate column into appropriate format
3. Use Conditional Split to compare the system variable VarMaxDate with ModifiedDate to identify the newly updated rows
4. Use Lookup to match the ID of new rows to existing rows
5. If the rows already exist, update the rows
6. Else add the rows to the table

6.4 DimSalesTerritory

6.4.1 Control Flow

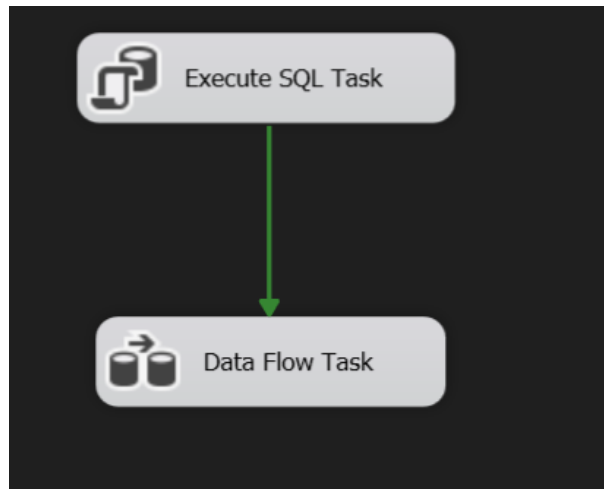


Figure 18: Control Flow for DimSalesTerritory

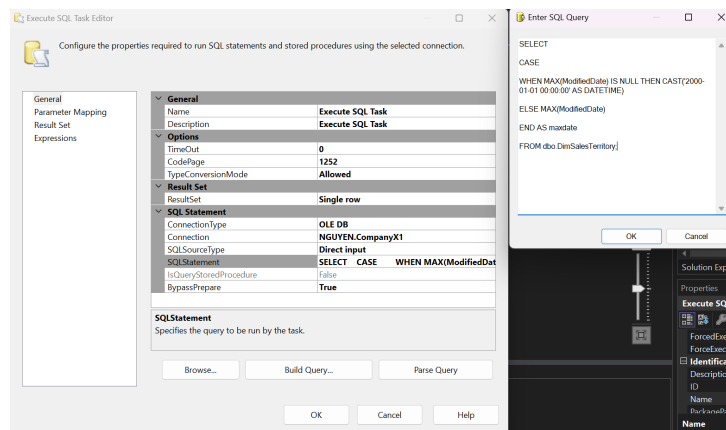


Figure 19: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the DimSalesTerritory and create a system variable VarMaxDate to use as comparison in the data flow.

6.4.2 Data Flow

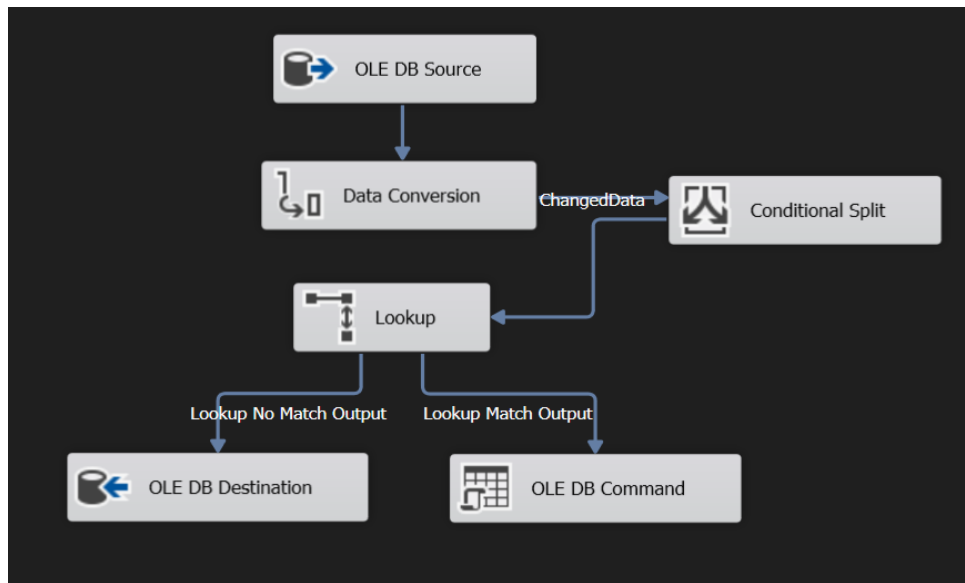


Figure 20: Data Flow for DimSalesTerritory

In general, the flow of data of DimSalesTerritory are as follow:

1. Select the columns in the schema by joining the following tables:
 - Person.CountryRegion
 - Sales.SalesTerritory
2. Convert the column ModifiedDate into the appropriate format
3. Use Conditional Split to compare the system variable VarMaxDate with ModifiedDate to identify the newly updated rows
4. Use Lookup to match the ID of new rows to existing rows
5. If the rows already exist, update the rows
6. Else add the rows to the table

6.5 DimPromotion

6.5.1 Control Flow

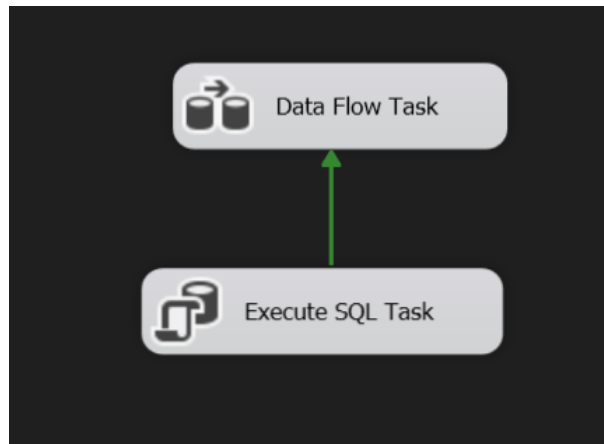


Figure 21: Control Flow for DimPromotion

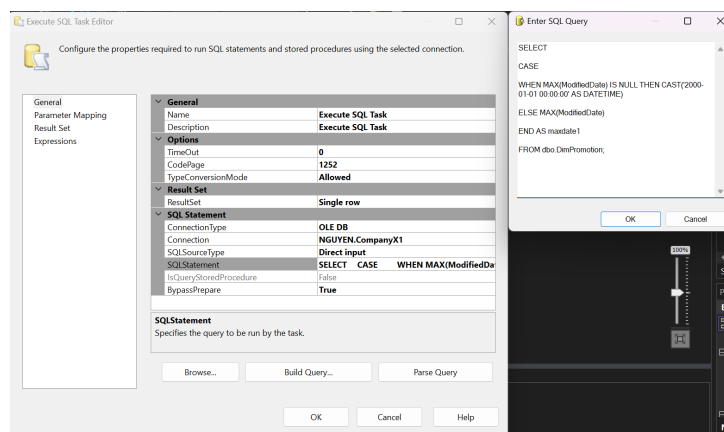


Figure 22: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the DimPromotion and create a system variable VarMaxDate to use as comparison in the data flow.

6.5.2 Data Flow

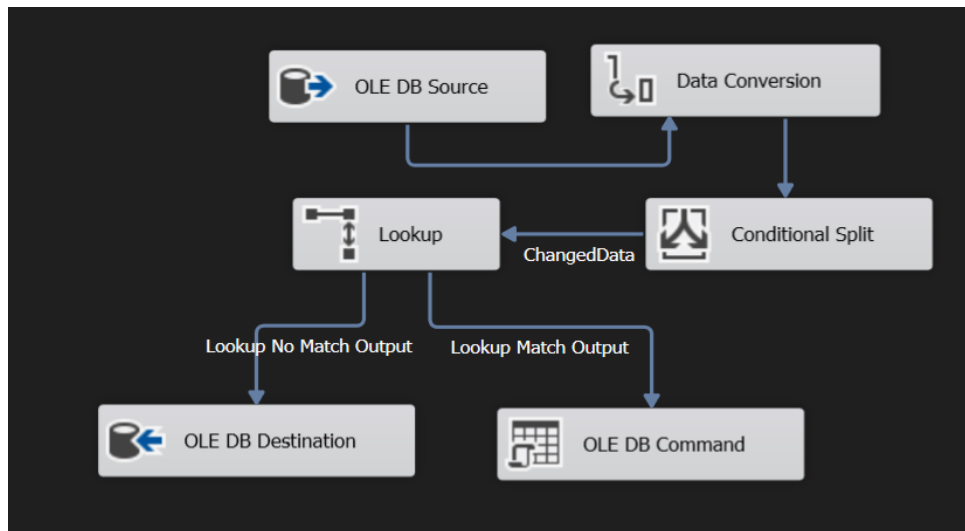


Figure 23: Data Flow for DimPromotion

In general, the flow of data of DimPromotion are as follow:

1. Select the columns in the schema from the table Sales.SpecialOffer
2. Convert the column ModifiedDate into the appropriate format
3. Use Conditional Split to compare the system variable VarMaxDate with ModifiedDate to identify the newly updated rows
4. Use Lookup to match the ID of new rows to existing rows
5. If the rows already exist, update the rows
6. Else add the rows to the table

6.6 FactSales

6.6.1 Control Flow

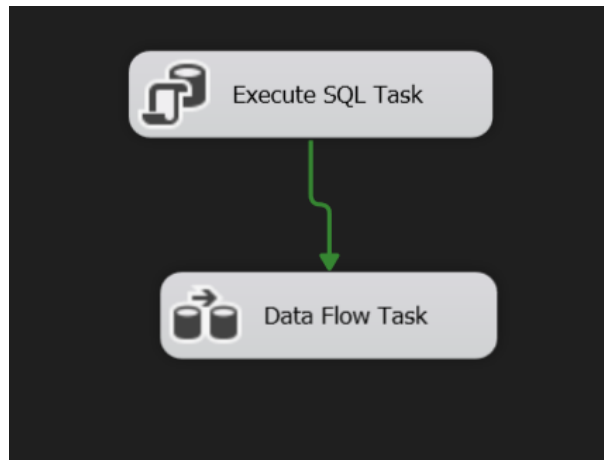


Figure 24: Control Flow for FactSales

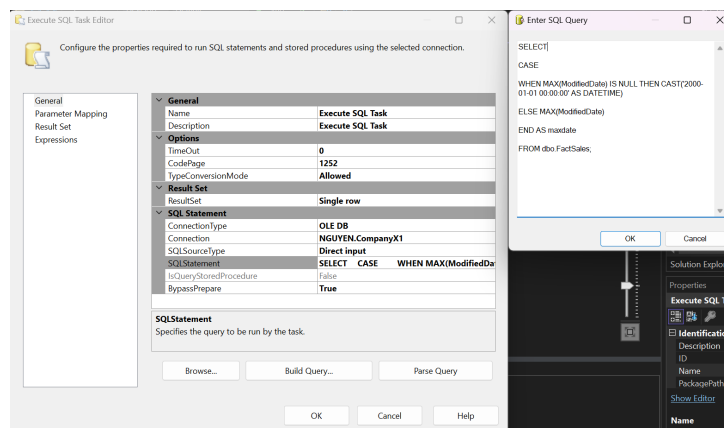


Figure 25: Settings for Execute SQL Task and query

In order to implement Incremental Loading, we need to have SQL task in order select the FactSales and create a system variable VarMaxDate to use as comparison in the data flow.

6.6.2 Data Flow

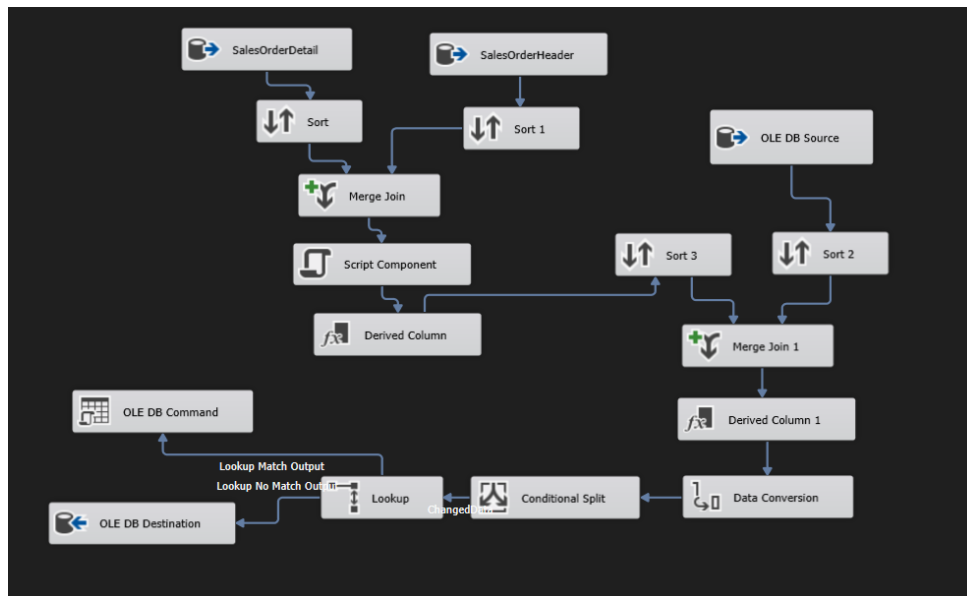


Figure 26: Data Flow for FactSales

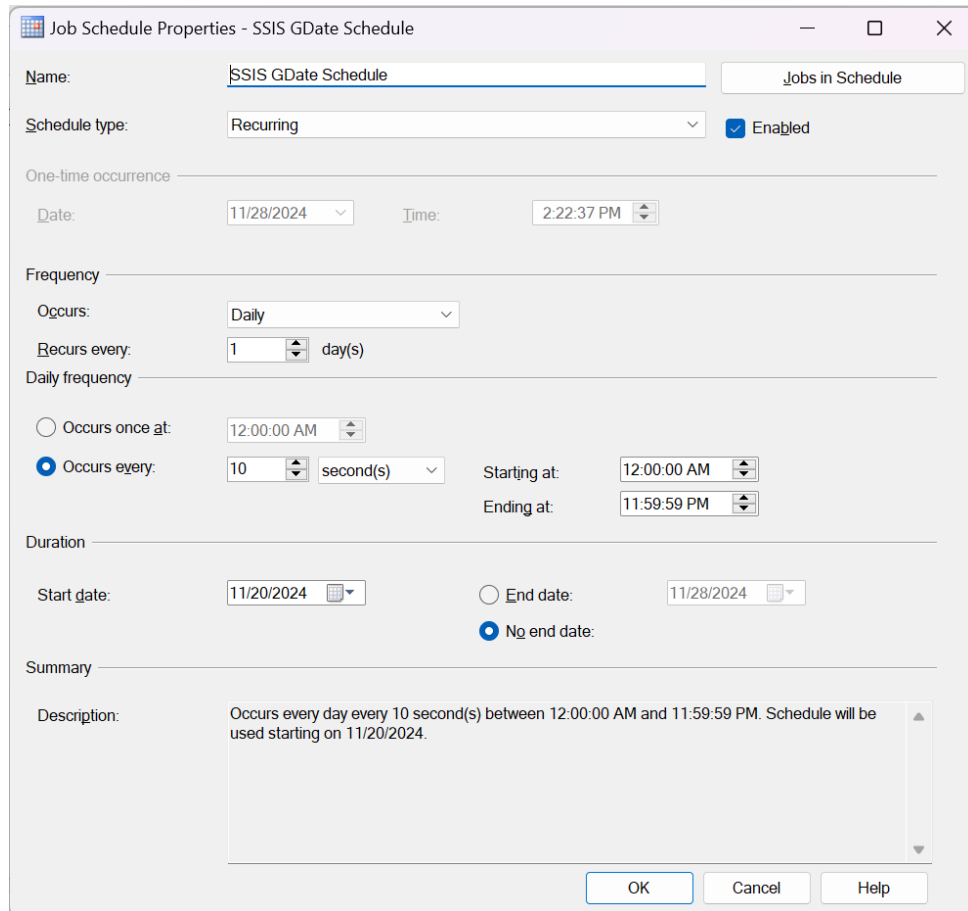
In general, the flow of data of FactSales are as follow:

1. Select the necessary columns from 3 tables which are:
 - Sales.SalesOrderHeader
 - Sales.SaleOrderDetail
 - Production.Product
2. Sort all three tables using the columns which will be used for merging which are:
 - SalesOrderID
 - ProductID
3. Merge Join the SalesOrderHeader and SalesOrderDetail using SalesOrderID
4. Use ScriptComponent to mark the multiple item in an order
5. Calculate TaxAmt and Freight using Derived Columnns
6. Merge join with the Product table using ProductID
7. Calculate TotalProductCost using Derived Columnn
8. Convert ModifiedDate to fit with the system
9. Use Conditional Split to compare the system variable VarMaxDate with ModifiedDate to identify the newly updated rows
10. Use Lookup to match the ID of new rows to existing rows

11. If the rows already exist, update the rows
12. Else add the rows to the table

6.7 Scheduling

Because every dimension we design is separated into different projects, we have to set up scheduling for each of our dimension tables, however, the scheduling will be the same. Here is how we set up the properties of our scheduler:



Job Schedule Properties - SSIS GDate Schedule

Name:

Schedule type: ☒ Enabled

One-time occurrence

Date: Time:

Frequency

Occurs:

Recurs every: day(s)

Daily frequency

☐ Occurs once at:

☒ Occurs every: second(s) Starting at:

Ending at:

Duration

Start date: ☐ End date:

☒ No end date:

Summary

Description:

Figure 27: Job Schedule Properties

This will be a recurring schedule which occurs every 10 seconds to achieve near real-time. Below will be a list containing all of the scheduling jobs that is available:










Agent Job Activity:										
	Name ^	Enabled	Status	Last Ru...	Last Run	Next Run	Category	Runnable	Schedul...	C...
	GCustomerSSIS	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GDateSSIS	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GDimProductSS...	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GFactSalesSSIS	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GGeographySSIS	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GPromotionSSIS	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	GSalesTerritory	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	SSIS Server Mai...	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0
	syspolicy_purge...	yes	Idle	Succee...	11/22/2...	not sch...	[Uncate...	yes	yes	0

Figure 28: Job Schedule

Below is a screenshots of all the successful scheduling jobs that has been done:

Log File Viewer - NGUYEN

Select Log: ☒ Job History ☐ Load Log ☐ Export ☐ Refresh ☐ Filter... Search...

Log file summary: No filter applied

Date*	Step ID	Server	Job Name	Step Name	Notifications	Message
11/22/2024 10:23:20 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:23:00 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:22:40 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:22:20 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:21:50 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:21:20 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:20:50 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:20:20 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:20:00 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:19:20 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:19:00 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:18:30 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:18:10 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:17:50 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:17:30 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:17:10 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:16:50 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:16:30 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:16:10 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:15:50 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:15:30 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:15:10 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:14:50 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:14:30 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:14:10 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:13:50 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).
11/22/2024 10:13:30 PM		NGUYEN	GdGateSSIS			The job succeeded.
11/22/2024 10:13:10 PM		NGUYEN	GdGateSSIS			The job was invoked by Schedule 12 (SSIS Gdgate Schedule).

Status: Last Refresh: 11/28/2024 2:20:55 PM

Filter Name: View file settings

Progress: Done (50 records)

Selected row details:

Date: 11/22/2024 10:23:20 PM
Log: Job History (GdGateSSIS)

Step ID: Server: NGUYEN Job Name: GdGateSSIS

Figure 29: Schedule Log File Viewer

7 Data Analysis

7.1 Data Extraction

From the Fact Sales table, We have derive into two aggregated dataset. The first one about information for each days per Sales Territory. And the second dataset is grouped each days per ProductCategoryName. We also bring along some features like OrderQty, UnitPrice, UnitPriceDiscount, TaxAmt, Freight for model training and prediction. We also calculate those aforementioned metrics to measures sales like total revenue, average profit margin, and average revenue per user.

Here is head of the aggregated datasets that group by OrderDate and TerritoryID:


```
1 df.head()
```

OrderDate	TerritoryID	TerritoryName	OrderQty	UnitPrice	UnitPriceDiscount	TaxAmt	Freight	SalesAmount	Average Profit Margin	Average_Revenue_Per_User
2011-05-31	1	Northwest, US	96.0	48231.2437	0.0	7784.447608	2432.639873	97305.5951	437.498446	13900.799300
2011-05-31	2	Northeast, US	35.0	12131.5067	0.0	1107.755176	346.173489	13846.9397	-128.293004	3461.734925
2011-05-31	3	Central, US	40.0	13886.5012	0.0	1539.445144	481.076606	19243.0643	174.893383	4810.766075
2011-05-31	4	Southwest, US	209.0	23173.7255	0.0	5580.654712	1743.954593	69758.1839	-4072.209214	9965.454843
2011-05-31	5	Southeast, US	126.0	60322.7235	0.0	10030.132072	3134.416266	125376.6509	529.687827	13930.738989

Figure 30: Head of Sales Metrics in everyday per Sales Territory

Here is head of the aggregated datasets that group by OrderDate and ProductCategoryName:

```
1 df.head()
```

OrderDate	ProductCategoryName	OrderQty	UnitPrice	UnitPriceDiscount	TaxAmt	Freight	TotalRevenue	Average Profit Margin	Average_Revenue_Per_User
2011-05-31	Accessories	72.0	464.2895	0.0	85.590760	26.747106	1069.8845	447.905061	82.298808
2011-05-31	Bikes	413.0	195999.4738	0.0	33106.679432	10345.837305	413833.4929	-2915.048050	10345.837322
2011-05-31	Clothing	148.0	715.0269	0.0	165.692768	51.778989	2071.1596	-906.125246	94.143618
2011-05-31	Components	102.0	14250.0435	0.0	2014.869968	629.646863	25185.8746	-1040.167617	1679.058307
2011-06-01	Accessories	0.0	0.0000	0.0	0.000000	0.000000	0.0000	0.000000	NaN

Figure 31: Head of Sales Metrics in everyday per Product Category

7.2 Date Preprocessing

For the first grouped dataset on OrderDate and TerritoryID, we have performed some steps like datatype correction and label encoding:

```
1 # Convert 'OrderDate' to datetime
2 df['OrderDate'] = pd.to_datetime(df['OrderDate'])
3
4 # Set 'OrderDate' as the index
5 df.set_index('OrderDate', inplace=True)
6
7 # Label Encoding TerritoryID into its Name
8 df['TerritoryID'] = df['TerritoryID'].astype('category')
9
10 territory_mapping = {
11     1: "Northwest, US",
12     2: "Northeast, US",
13     3: "Central, US",
14     4: "Southwest, US",
15     5: "Southeast, US",
16     6: "Canada",
17     7: "France",
18     8: "Germany",
19     9: "Australia",
20     10: "United Kingdom"
21 }
22
23 df['TerritoryName'] = df['TerritoryID'].map(territory_mapping)
24
25 df.insert(1, 'TerritoryName', df.pop('TerritoryName'))
```

With the second aggregated dataset on OrderDate and ProductCategoryName, we also made necessary datatype corrections and and remove NaNs Category:

```
1 # Convert 'OrderDate' to datetime
2 df['OrderDate'] = pd.to_datetime(df['OrderDate'])
3
4 # Set 'OrderDate' as the index
5 df.set_index('OrderDate', inplace=True)
6
7 # Remove NaNs values
8 df = df.dropna()
```

We also drop metric Average Revenue Per User since it contain too many NaNs value:

```
1 non_nan_count = df['Average_Revenue_Per_User'].count()
2 print(f"Number of non-NaN values in 'Average_Revenue_Per_User': {non_nan_count}"
3       )
4 nan_count = df['Average_Revenue_Per_User'].isna().sum()
5 print("\nNumber of NaN values in 'Average_Revenue_Per_User':", nan_count)
```

Output:

Number of non-NaN values in 'Average_Revenue_Per_User': 1977

Number of NaN values in 'Average_Revenue_Per_User': 3643

7.3 Visualization and Key insights

7.3.1 Datasets that group by OrderDate and TerritoryID

Below are a list of visualizations and key findings we have found:

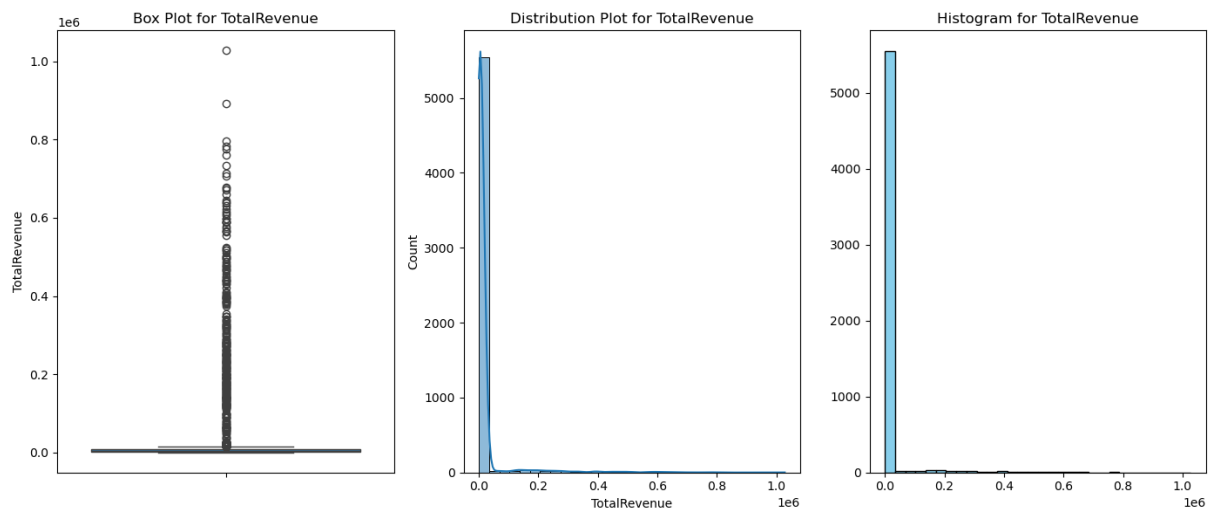


Figure 32: Total Revenue

- The box plot highlights significant outliers, with extreme values reaching over 1,000,000, while most data points are concentrated around zero.

- The distribution and histogram plots show a heavily right-skewed distribution, with the majority of transactions having low sales amounts, indicating occasional high-value transactions dominate the upper range.
- This suggests the need for normalization or log transformation before applying predictive models or statistical analysis to address skewness.

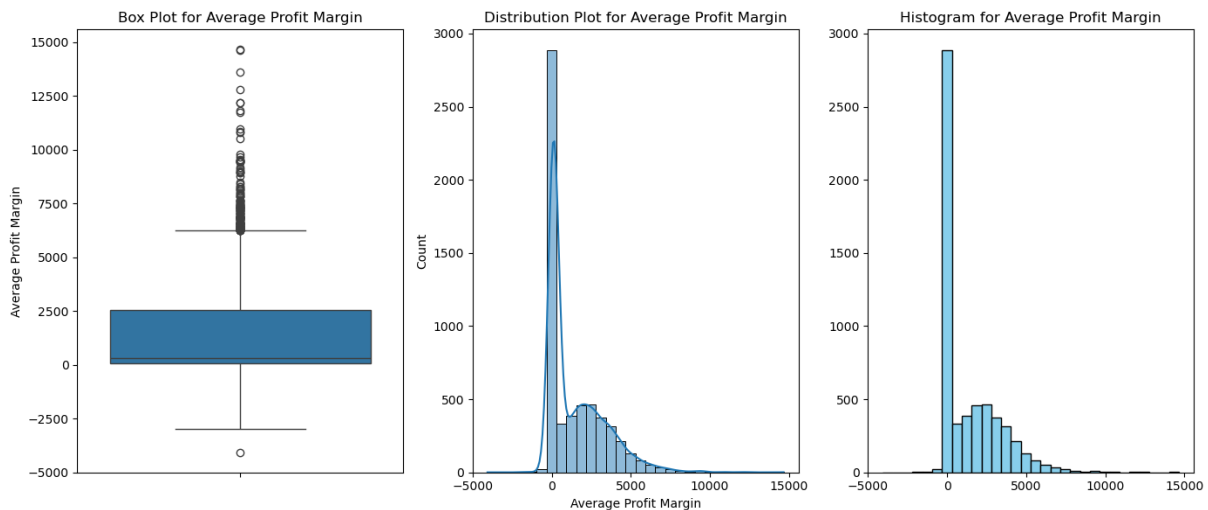


Figure 33: Average Profit Margin

- The box plot reveals a wide range of profit margins with numerous outliers, both positive and negative, suggesting variability in transaction profitability.
- The distribution plot shows a peak around zero and a long right tail, indicating most transactions are either low or moderately profitable, with fewer cases of very high profit margins.
- Negative values in the histogram highlight that some transactions result in losses, which need further investigation into their causes.

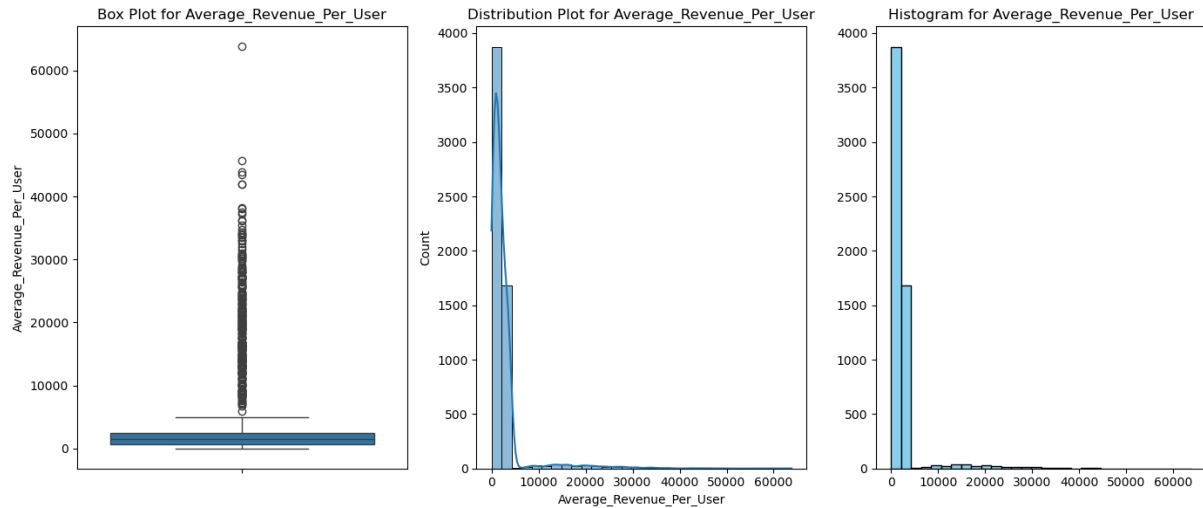


Figure 34: Average Revenue Per User

- The box plot demonstrates the presence of substantial outliers, with a small number of average users contributing significantly higher revenue than the majority.
- The distribution and histogram plots show a steeply right-skewed pattern, where the vast majority of users contribute relatively low revenue.
- The large concentration near zero implies the dataset may have a few high-value users that drive revenue, emphasizing the importance of segmentation for targeted strategies.

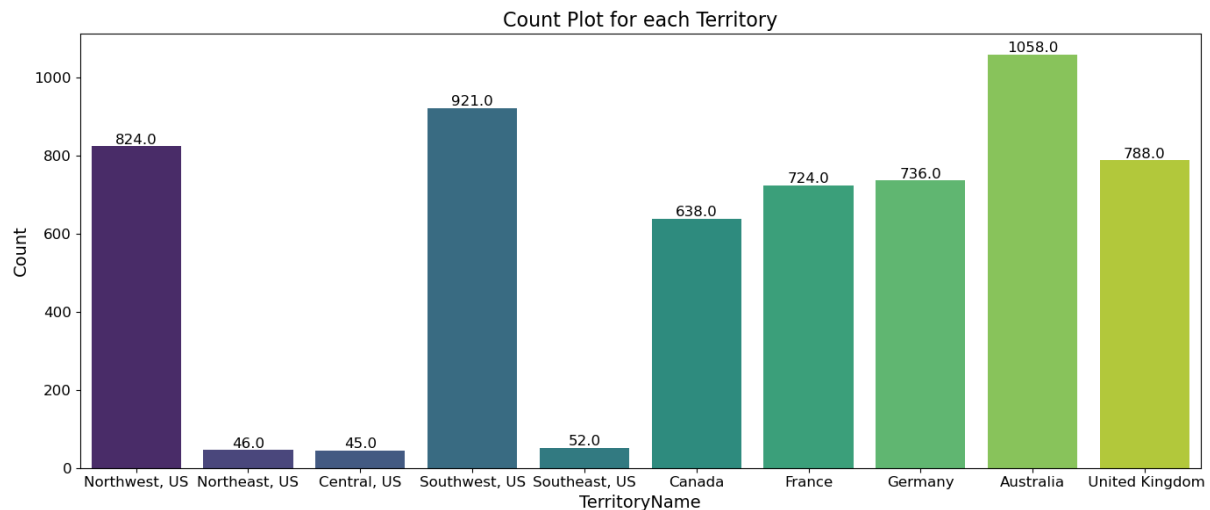


Figure 35: Count Plot per Sales Territory

- The count plot shows that Australia has the highest number of transactions among all territories, followed by Southwest US and Northwest US.
- In contrast, the Northeast, Southeast, and Central US have significantly lower transaction counts, indicating potential regional disparities in sales activities.

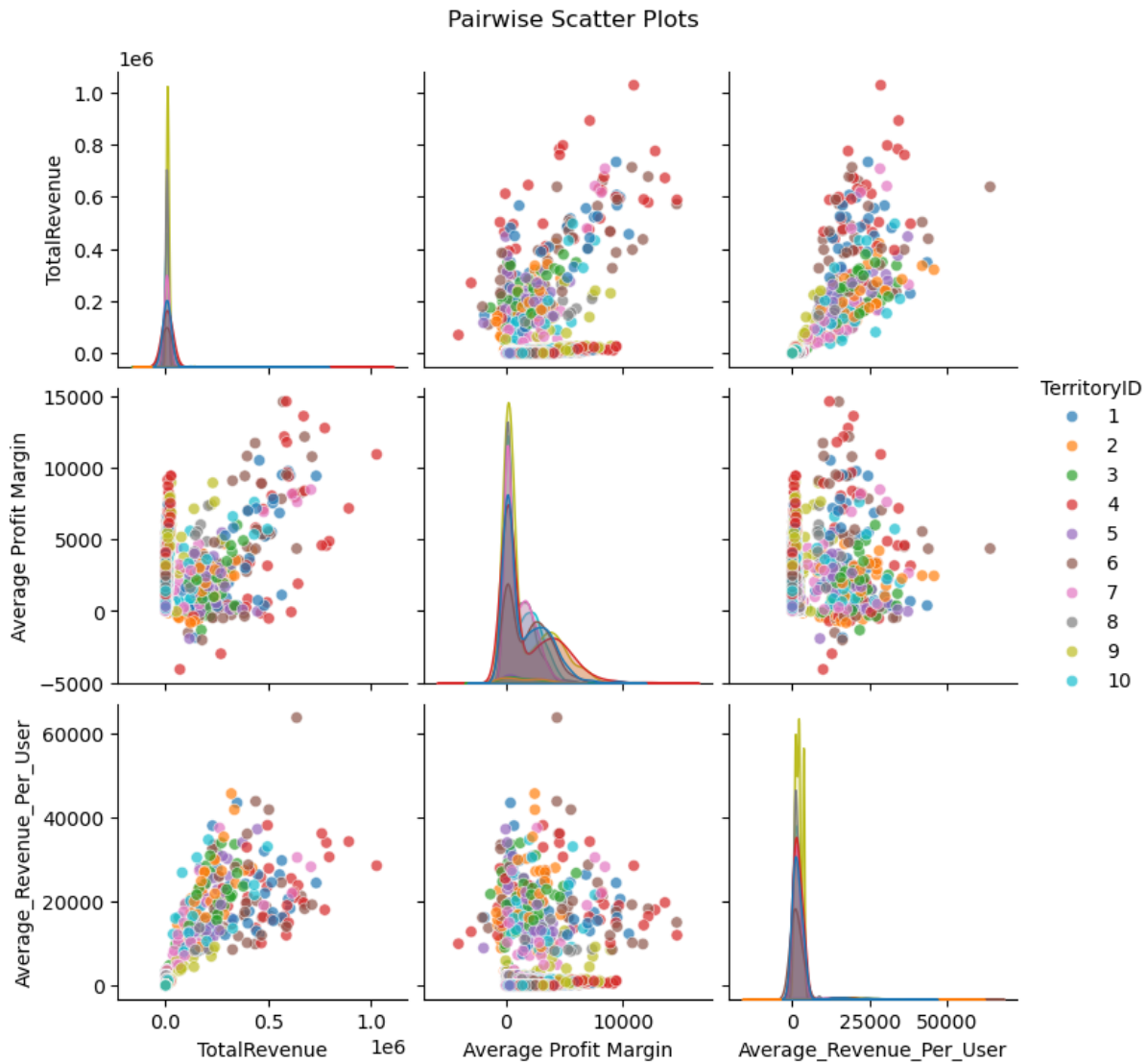


Figure 36: Pairwise Scatter Plots of some Features

- The pairwise scatter plot highlights strong positive relationships between Total Revenue and Average Profit Margin, as well as between Total Revenue and Average Revenue Per User, suggesting that higher total revenue tends to accompany higher profitability and per-user revenue.
- However, there is noticeable variability among different territories, with some regions showing outliers that achieve exceptionally high values in these metrics.
- Density plot on diagonal line reveal that most values of these metrics tend to focus on lower ranges, indicating that only a few regions or transactions drive significantly higher sales performance.

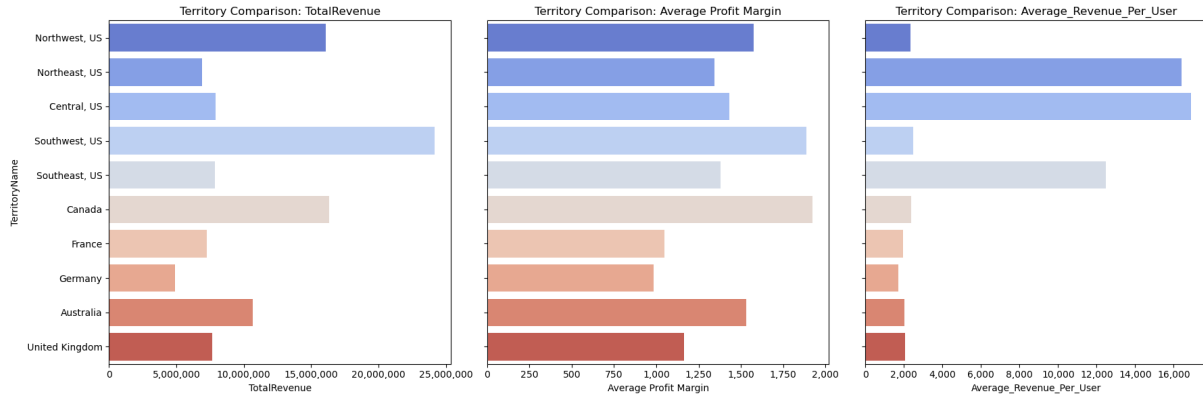


Figure 37: 3 Sales Metrics for each Sales Territories

- **Total Revenue:** The Southwest US leads with approximately 24 million in total revenue, followed by Canada and Northwest US at 16 million each. In contrast, Germany has a lowest total with 4.9 million.
- **Average Profit Margin:** Canada has the highest average profit margin, closely followed by the Southwest US, indicating strong profitability. Germany and France have the lowest margins, suggesting possible operational inefficiencies or pricing challenges.
- **Average Revenue Per User:** Northeast, Central, and Southeast US lead with over 12000, reflecting valuable customer bases. The Southwest US, despite having high total revenue and profit margin, having the least per-user revenue, indicating reliance on a larger customer base with smaller contributions.
- **Overall:** The Southwest US and Canada excel in total revenue and profit margin, reflecting active markets. France and Germany consistently rank lowest across all metrics, indicating issues like limited market penetration or weaker customer spending. Strategies to boost engagement, optimize pricing, or reduce costs could implied to improve performance in these regions.

For every line chart, we resample values of those sales metrics into quarterly to make the line smoother.

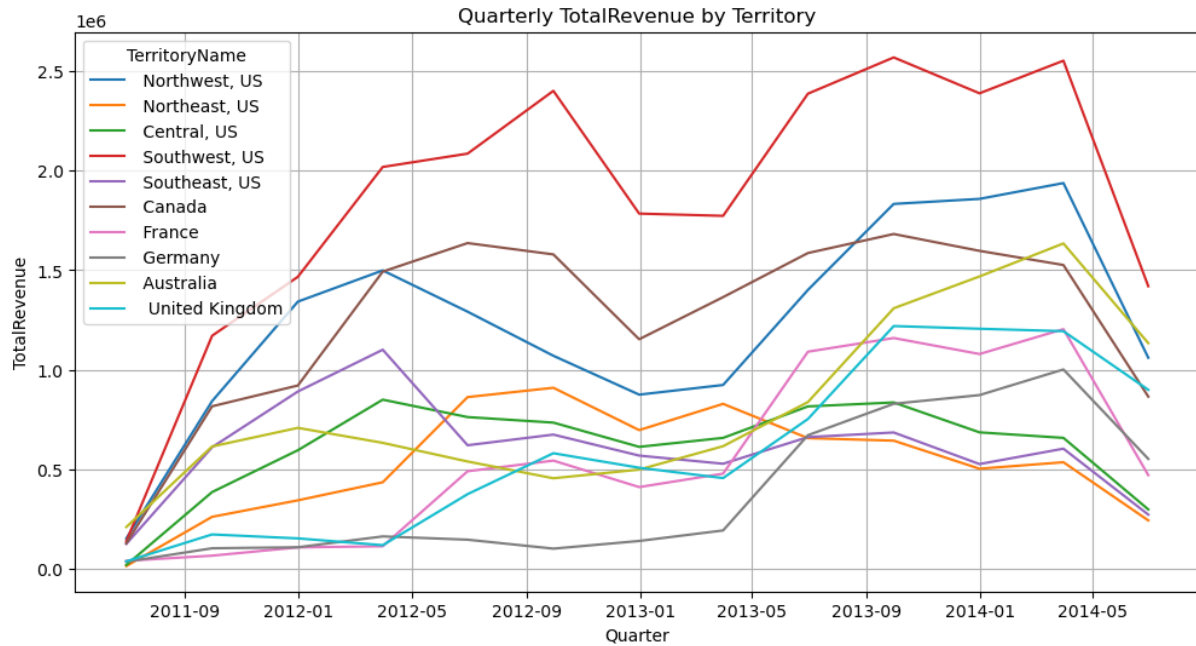


Figure 38: Quarterly Total Revenue by Territory

- As can be seen in the line chart, Southwest US like the bar plot above consistently outperforms other regions in total revenue, with notable peaks in late 2013 and early 2014, reaching over 2.5 millions.
- Other territories like the Northwest US, Canada, and Australia show steady growth but remain significant behind the Southwest US
- Smaller territories like Southeast and Northeast US contribute minimal revenue to the whole, indicating regional disparities
- Total Revenue across all Territories seem to fall simultaneously around April-2014, thus may be due to financial crisis or some other reasons.

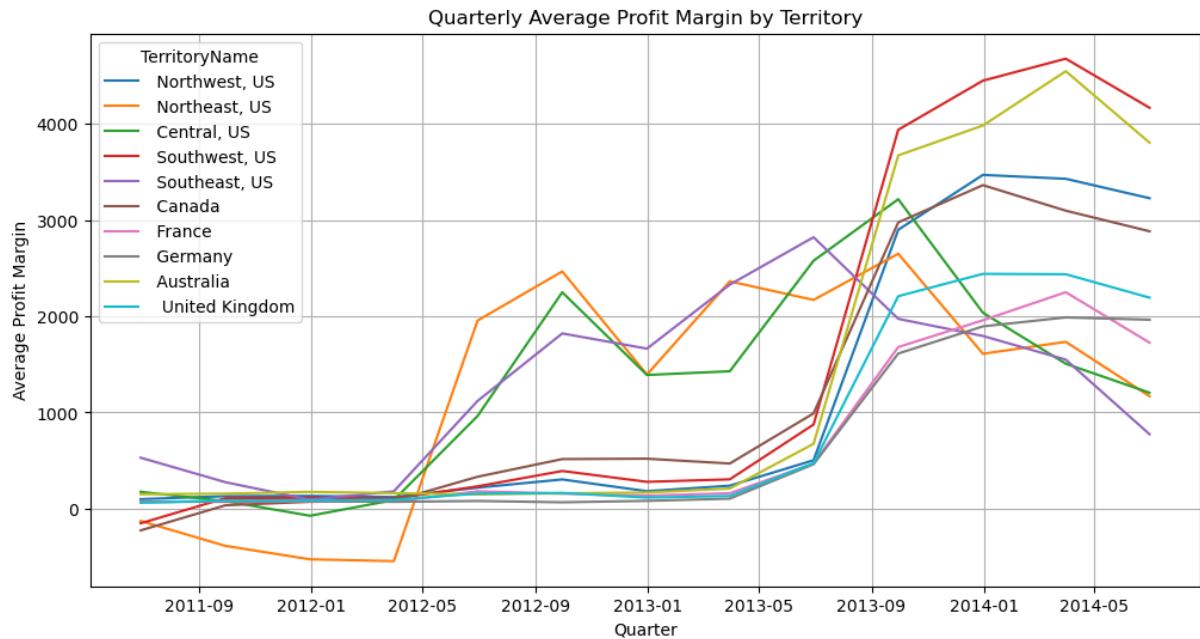


Figure 39: Quarterly Average Profit Margin by Territory

- In the line chart, profit margins across territories show a steep increase from late 2011 to mid-2013, especially in the Southwest US and Australia.
- By 2014, most regions experience a decline in profit margins, though the Southwest US and Australia maintain comparatively higher levels.
- Territories such as Germany and France exhibit modest growth, while Northeast US remains stagnant with low margins.

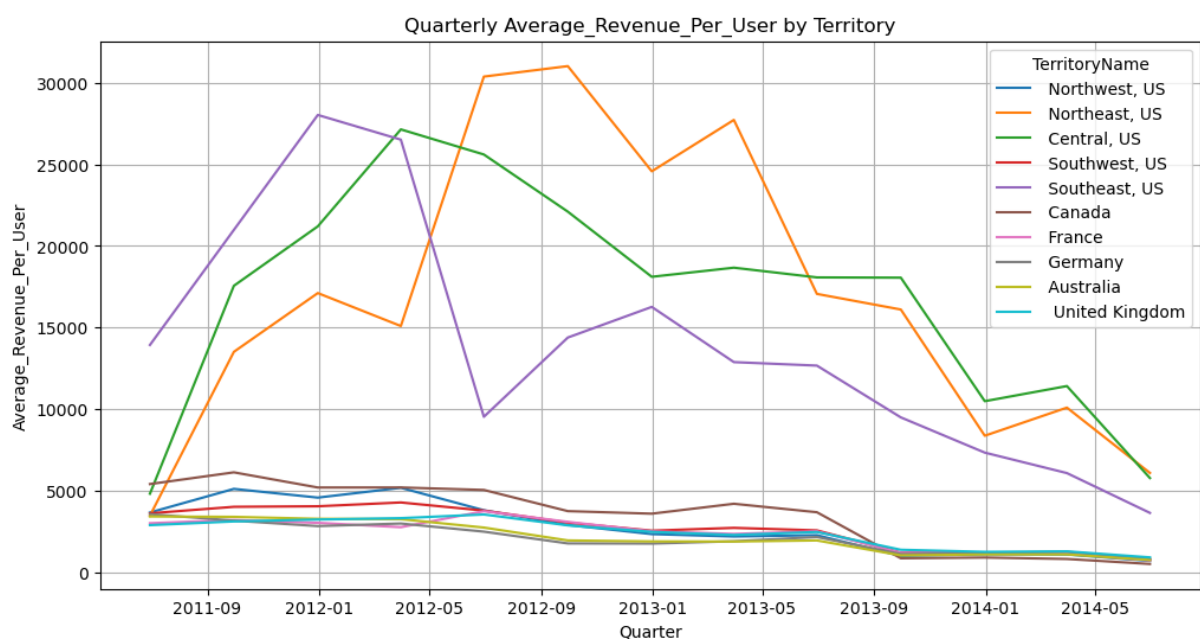


Figure 40: Quarterly Average Revenue per User by Territory

- From the line, Northeast, Southeast, and Central US always have high average revenue per user, separate from other Territories Regions but these line fluctuated a lot with a deep downward trend between the data range.
- Other Territories line seem to exhibit consistent small downward trends and always stick to each other throughout the period.

7.3.2 Datasets that group by OrderDate and ProductCategoryName

Below are some visualizations and insights concluded:

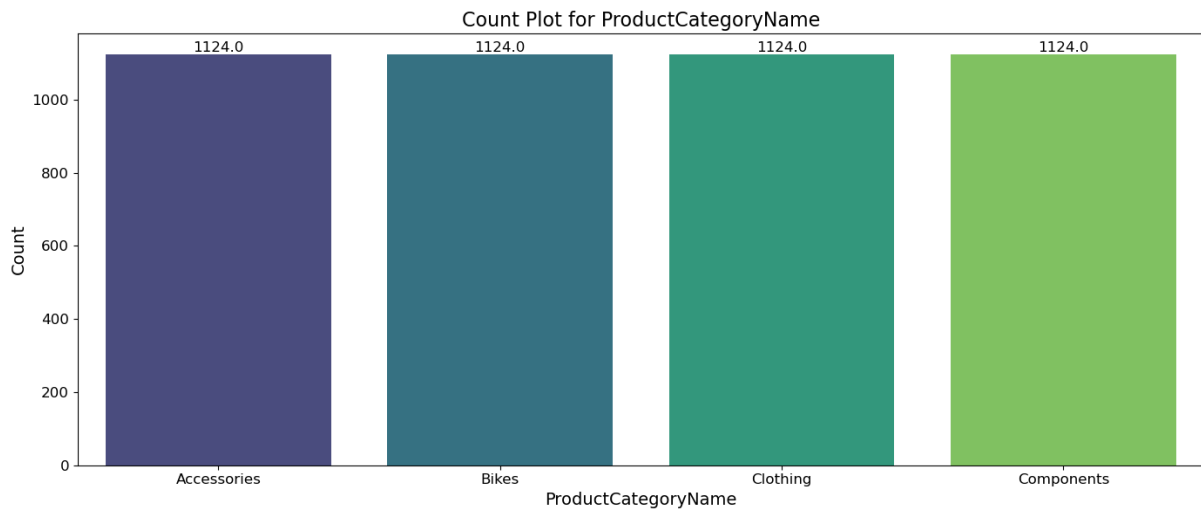


Figure 41: Frequencies of each Product Category

- From the count plot, we can see that each type of product has equal appearance in 1124 days.
- However, the dataset contains 1127 dates between 2011-05-31 and 2014-06-30, which means there exist 3 dates that the product don't have value

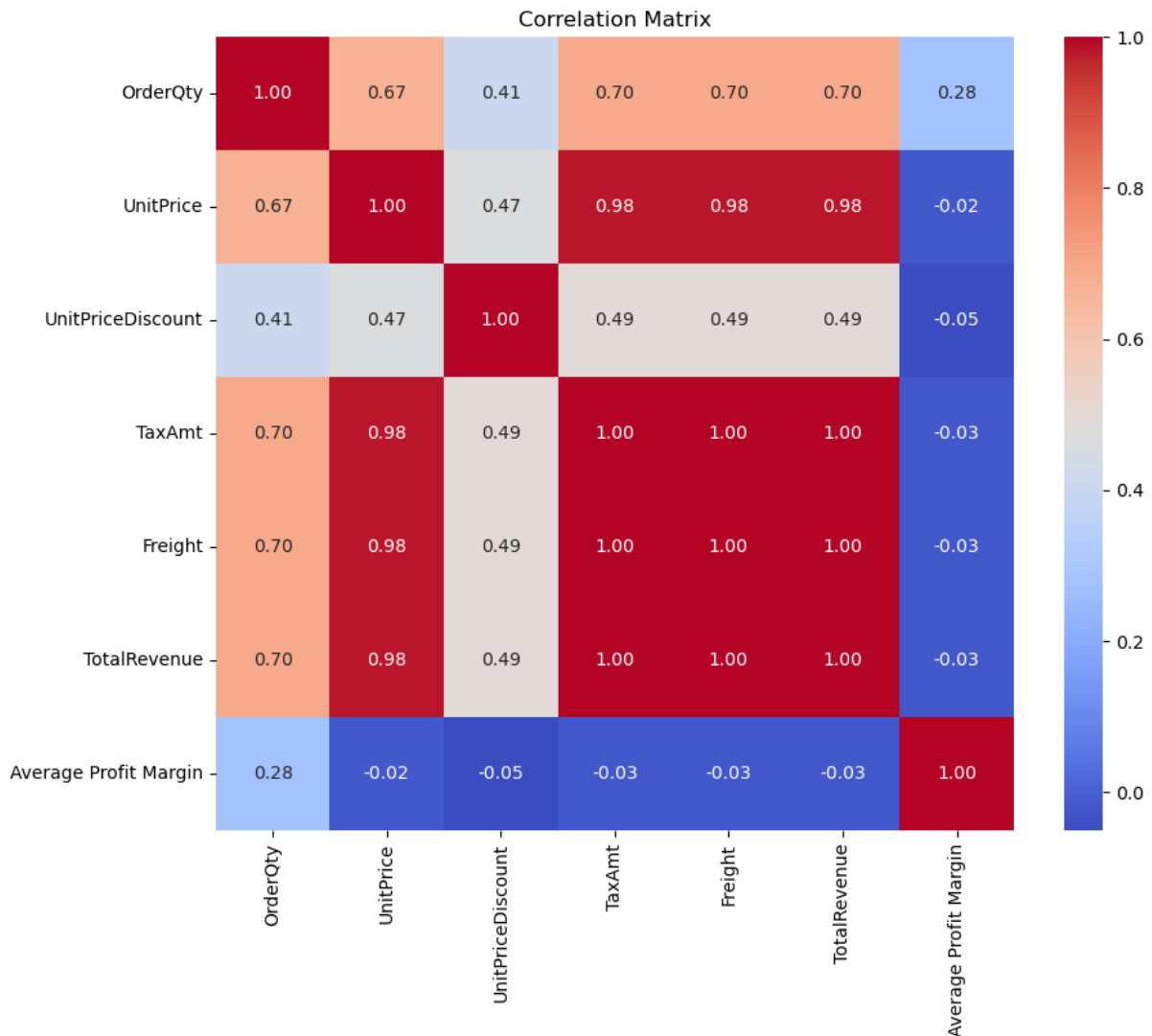


Figure 42: Heatmap of each feature

- As seen from the heatmap, UnitPrice, TaxAmt, Freight, and TotalRevenue exhibit near-perfect positive correlations (0.98 to 1.00), indicating that these metrics increase proportionally. This reflects the natural relationship in aggregated sales data where higher-priced orders contribute to higher taxes, freight costs, and total revenue.
- Average Profit Margin shows very weak or slightly negative correlations with all other metrics (ranging from -0.05 to 0.28). This indicates that higher revenues, order quantities, or prices do not necessarily lead to higher profit margins and may reflect other factors.

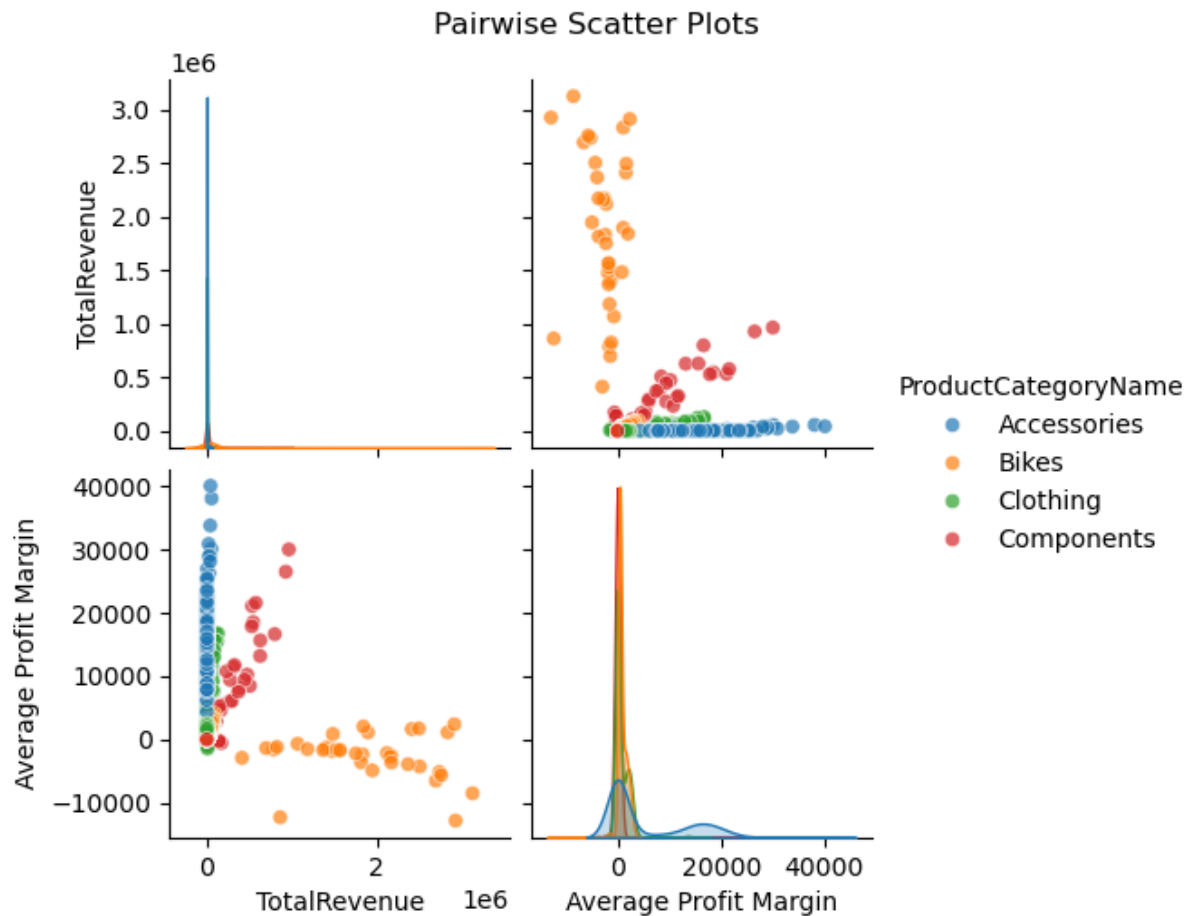


Figure 43: Pairwise Scatter Plots for 2 sales metrics

- With the scatter plots, we can see that Bikes generate the highest TotalRevenue, reaching up to 3 million, but often come with a lower Average Profit Margin, including some negative margins, likely due to high costs or discounts.
- Accessories and Components contribute moderate revenue but achieve higher profit margins, indicating more consistent profitability.
- Clothing show lower total revenue and profit margins, suggesting they are less significant contributors to overall sales performance.

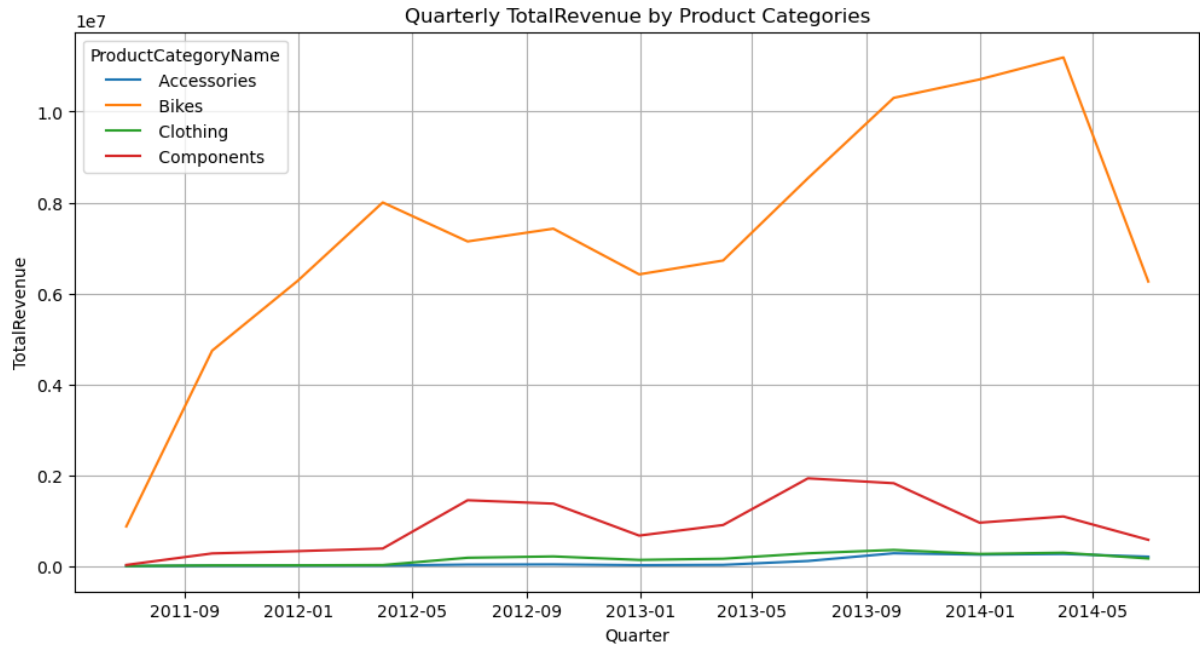


Figure 44: Line Chart for Total Revenue per Product Category

- Bikes dominate total revenue, experiencing consistent growth throughout the date range, peaking above 10 million before a sharp decline in April 2014.
- Components show moderate growth, peaking around mid 2013 with 2 million, but remain significantly lower than Bikes.
- Other categories, including Accessories, Clothing, contribute marginally to total revenue, indicating they are less critical to overall sales performance.

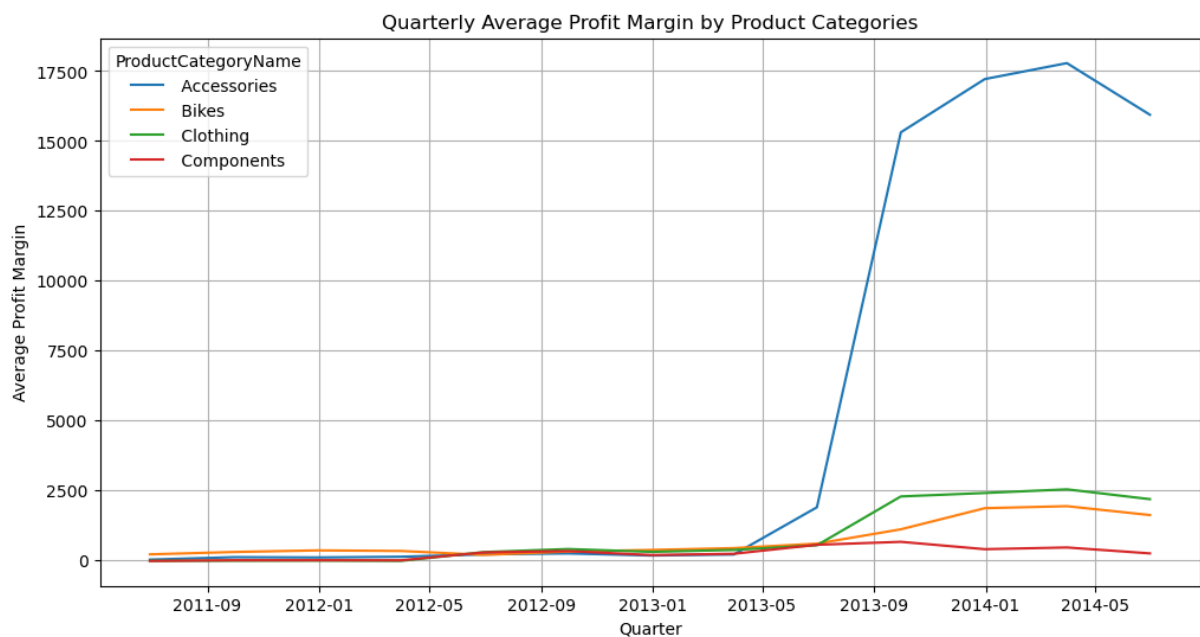


Figure 45: Line Chart for Average Profit Margin per Product Category

- Accessories exhibit a dramatic increase in average profit margin starting in mid 2013, peaking above 17500, far passing all other categories.
- Clothing and Bikes show a steady increase in profit margin in a similar date range as Accessories.
- Components maintain low and relatively flat profit margins, indicating limited profitability growth in these categories.

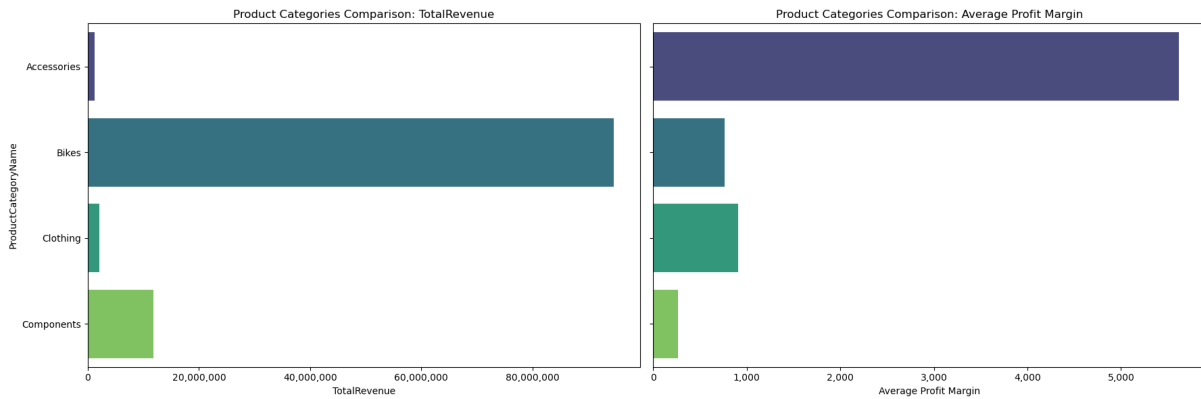


Figure 46: Bar Plot for 2 Metrics per Product Category

- **Total Revenue:** The Bikes category leads significantly, generating nearly 100 million in total revenue. Components and Clothing contribute moderately, while Accessories has little to no contribution in total revenue, indicating a smaller market share.
- **Average Profit Margin:** Accessories dominate with the highest average profit margin, far surpassing other categories. Bikes, Clothing, and Components show moderate profit margins.
- **Overall Analysis:** The Bikes category clearly dominates in total revenue, indicating a high-volume sales driver, though its profit margins are moderate. In contrast, Accessories excel in profitability despite generating minimal revenue, suggesting high-margin, low-volume sales. Components and Clothing perform moderately in both metrics. These findings suggest focusing on optimizing sales strategies for Bikes while leveraging the profitability of Accessories to maximize overall business performance.

7.4 Hypothesis Testing

Hypothesis Question: Are there significant differences in any of the three key sales metrics (Average Profit Margin, Average Revenue Per User, or Total Revenue) between territories?

Formal Hypotheses:

- **Null Hypothesis (H_0):** The means of all three sales metrics are the same across all territories.

- **Alternative Hypothesis (H_1):** The mean of at least one of the sales metrics differs for at least one territory.

Test to Use: We use Welch's ANOVA test due to several reasons related to nature of the grouped dataset:

1. Large differences between Territories Size.
2. The Variance of all 3 sales metrics differs significantly between territories.
3. From the boxplot, the dataset contains many outliers.
4. With large sample size, even small deviations from assumptions can lead to misleading results.

```
1 import pingouin as pg
2
3 # Perform Welch's ANOVA for each metric
4 welch_anova1 = pg.welch_anova(dv='Average Profit Margin', between='TerritoryName', data=df)
5 print(welch_anova1)
6
7 welch_anova2 = pg.welch_anova(dv='Average Revenue Per User', between='TerritoryName', data=df)
8 print(welch_anova2)
9
10 welch_anova3 = pg.welch_anova(dv='TotalRevenue', between='TerritoryName', data=df)
11 print(welch_anova3)
12
13 # Extract the p-values
14 p_value1 = welch_anova1['p-unc'][0]
15 p_value2 = welch_anova2['p-unc'][0]
16 p_value3 = welch_anova3['p-unc'][0]
17
18 # Apply Bonferroni correction for multiple comparisons
19 alpha = 0.05
20 corrected_alpha = alpha / 3 # Bonferroni correction for 3 tests
21
22 # Conclusion based on the corrected p-values
23 if (p_value1 < corrected_alpha) & (p_value2 < corrected_alpha) & (p_value3 < corrected_alpha):
24     print("\nConclusion: All three p-values are less than the corrected alpha (0.05/3).")
25     print("There are significant differences in all three sales metrics between territories.")
26 elif (p_value1 < corrected_alpha) | (p_value2 < corrected_alpha) | (p_value3 < corrected_alpha):
27     print("\nConclusion: At least one p-value is less than the corrected alpha (0.05/3).")
```

```

28     print("There are significant differences in at least one of the sales
        metrics between territories.")
29 else:
30     print("\nConclusion: None of the p-values are less than the corrected alpha
        (0.05/3).")
31     print("There are no significant differences in the three sales metrics
        between territories.")

```

Output:

	Source	ddof1	ddof2	F	p-unc	np2
0	TerritoryName	9	471.185864	26.073329	1.736547e-36	0.034054
	Source	ddof1	ddof2	F	p-unc	np2
0	TerritoryName	9	461.274227	28.288021	4.340132e-39	0.219155
	Source	ddof1	ddof2	F	p-unc	np2
0	TerritoryName	9	461.508441	32.019794	1.437049e-43	0.104154

Conclusion: All three p-values are less than the corrected alpha (0.05/3).
There are significant differences in all three sales metrics between territories.

7.5 Regression Analysis

For the regression analysis, we group the dataset only by OrderDate to predict 3 Sales Metrics TotalRevenue, Average Profit Margin, and Average_Revenue_Per_User

7.5.1 Data Preparation and Preprocessing

- **Dataset Loading:** The dataset regression_dataset.csv was loaded, and the OrderDate column was converted to a datetime format before setting it as the index.
- **Reindexing:** The dataset was then reindexed to cover all dates within the range, filling missing values using forward (ffill) to maintain continuity.
- **Winsorization:** To handle outliers, each feature was winsorized between 1st and 99th percentiles.
- **Feature and Target Selection:**
 - **Features:** OrderQty, UnitPrice, UnitPriceDiscount, TaxAmt, and Freight.
 - **Targets:** TotalRevenue, Average Profit Margin, and Average_Revenue_Per_User.
- **Normalization** The StandardScaler was used to normalize both features and targets to ensure consistent scaling for regression models.

Code:

```
1 df = pd.read_csv('regression_dataset.csv')
2 # Convert 'OrderDate' to datetime
3 df['OrderDate'] = pd.to_datetime(df['OrderDate'])
4
5 # Set 'OrderDate' as the index
6 df.set_index('OrderDate', inplace=True)
7 all_dates = pd.date_range(df.index.min(), df.index.max(), freq='D')
8
9 # Step 2: Reindex the DataFrame to include all dates in the full range
10 df = df.reindex(all_dates)
11
12 # Step 3: Fill NaN values
13 df.ffill(inplace=True) # Replace 0 with the desired fill value or method
14 all_dates = pd.date_range(df.index.min(), df.index.max(), freq='D')
15
16 missing_date = set(all_dates) - set(df.index.unique())
17
18 def winsorize(series, lower=0.01, upper=0.99):
19     lower_limit = series.quantile(lower)
20     upper_limit = series.quantile(upper)
21     return series.clip(lower=lower_limit, upper=upper_limit)
22
23 for column in df.columns:
24     df[column] = winsorize(df[column])
25
26 # Define features and targets
27 features = ['OrderQty', 'UnitPrice', 'UnitPriceDiscount', 'TaxAmt', 'Freight']
28 targets = ['TotalRevenue', 'Average Profit Margin', 'Average_Revenue_Per_User']
29
30 # Normalize features
31 scaler_X = StandardScaler()
32 X_scaled = scaler_X.fit_transform(df[features])
33
34 # Normalize targets
35 scalers_y = {target: StandardScaler() for target in targets}
36 y_scaled = {target: scalers_y[target].fit_transform(df[[target]]) for target in targets}
```

7.5.2 Model Selection and Hyperparameter Tuning

Three regression models were selected for analysis:

- Random Forest Regressor
- Gradient Boosting Regressor
- Ridge Regression

Each model underwent hyperparameter tuning using **GridSearchCV** with a **TimeSeriesSplit**. The best model for each target was chosen based on the lowest **negative mean squared error (MSE)**

Code:

```
1 # Split data using TimeSeriesSplit
2 tscv = TimeSeriesSplit(n_splits=3)
3 splits = list(tscv.split(X_scaled))
4
5 # Define model and hyperparameter grid
6 param_grid_rf = {
7     'model__n_estimators': [50, 100, 200],
8     'model__max_depth': [3, 5, 7],
9     'model__min_samples_split': [2, 5],
10    'model__min_samples_leaf': [1, 2]
11 }
12 param_grid_gb = {
13     'model__n_estimators': [50, 100, 200],
14     'model__learning_rate': [0.01, 0.1, 0.2],
15     'model__max_depth': [3, 5, 7]
16 }
17 param_grid_ridge = {'model__alpha': [0.1, 1, 10]}
18
19 models = {
20     "Random Forest": (RandomForestRegressor(random_state=42), param_grid_rf),
21     "Gradient Boosting": (GradientBoostingRegressor(loss='huber', random_state
22                          =42), param_grid_gb),
23     "Ridge Regression": (Ridge(), param_grid_ridge)
24 }
25
26 # Hyperparameter tuning
27 best_models = {}
28 for target in targets:
29     print(f"Hyperparameter tuning for target: {target}")
30     y_target = y_scaled[target].ravel()
31
32     for model_name, (model, param_grid) in models.items():
33         print(f"    Tuning {model_name}")
34         pipeline = Pipeline([('scaler', StandardScaler()), ('model', model)])
35         grid_search = GridSearchCV(
36             pipeline,
37             param_grid,
38             scoring='neg_mean_squared_error',
39             cv=[(train_index, test_index) for train_index, test_index in splits
40                 ],
41             n_jobs=-1
42         )
43         grid_search.fit(X_scaled, y_target)
44
45         # Save best model
46         best_models[f"{target}_{model_name}"] = grid_search.best_estimator_
47         print(f"    Best parameters for {target} with {model_name}: {grid_search
48               .best_params_}")
```

7.5.3 Evaluation Metrics

The models were evaluated using the following metrics:

- Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
- Mean Absolute Error (MAE): Measures the average absolute difference between actual and predicted values.
- R² Score: Represents the proportion of variance explained by the model.

Evaluation Conclusions:

- **TotalRevenue:** Ridge Regression performed the best with an MSE of 0.0021, MAE of 0.0142, and an R² of 0.9981, indicating excellent accuracy and minimal error. This model is the most reliable for predicting sales amounts.
- **Average Profit Margin:** Random Forest is the best-performing model with an MSE of 0.3815, MAE of 0.2537, and an R² of 0.2132. Although the R² values are low for all models, Random Forest captures the trends better than Gradient Boosting and Ridge Regression.
- **Average_Revenue_Per_User:** Gradient Boosting achieved the best results with an MSE of 0.1008, MAE of 0.1468, and an R² of 0.8584, indicating strong predictive performance and accuracy for this metric.

Overall, Ridge Regression excels for predicting TotalRevenue, Random Forest performs best for Average Profit Margin, and Gradient Boosting is the top choice for Average_Revenue_Per_User.

```
1 # Evaluation loop
2 results = {}
3 for target in targets:
4     print(f"\nTraining and evaluating models for target: {target}\n")
5     y_target = y_scaled[target].ravel()
6
7     target_results = {"MSE": {}, "MAE": {}, "R2": {}}
8     for model_name, (model, _) in models.items(): # Extract only the model from
9         # the tuple
10         mse_splits, mae_splits, r2_splits = [], [], []
11         for train_index, test_index in tscv.split(X_scaled):
12             X_train, X_val = X_scaled[train_index], X_scaled[test_index]
13             y_train, y_val = y_target[train_index], y_target[test_index]
14
15             # Train model
16             model.fit(X_train, y_train)
17
18             # Predict on validation data (scaled domain)
19             y_pred_val = model.predict(X_val)
```

```
20         # Calculate metrics in scaled domain
21         mse_splits.append(mean_squared_error(y_val, y_pred_val))
22         mae_splits.append(mean_absolute_error(y_val, y_pred_val))
23         r2_splits.append(r2_score(y_val, y_pred_val))
24
25     # Average results across splits
26     target_results["MSE"][model_name] = np.mean(mse_splits)
27     target_results["MAE"][model_name] = np.mean(mae_splits)
28     target_results["R2"][model_name] = np.mean(r2_splits)
29
30     print(f"{model_name} - MSE: {target_results['MSE'][model_name]:.4f}, "
31           f"MAE: {target_results['MAE'][model_name]:.4f}, "
32           f"R2: {target_results['R2'][model_name]:.4f}")
33
34     results[target] = target_results
```

7.5.4 Final Predictions and Model Performance

The best models for each target were used to make final predictions on the last test split. Predictions were inverse-transformed to the original scale, and the models' performance was evaluated:

- MSE: Measures the accuracy of the final predictions.
- MAE: Indicates the average error magnitude.
- R²: Reflects how well the model fits the data.

Actual vs. Predicted Plot: Scatter plots of actual vs. predicted values indicated that most predictions aligned closely with the perfect prediction line, suggesting strong model performance.

```
1 # Final prediction and inverse transform
2 for target in targets:
3     # Retrieve the best model for the target
4     model = None
5     for key, saved_model in best_models.items():
6         if key.startswith(target): # Match the target in the key
7             model = saved_model
8             break
9
10    if model:
11        y_test_scaled = y_scaled[target][splits[-1][1]]
12        y_pred_test_scaled = model.predict(X_scaled[splits[-1][1]])
13
14        # Inverse transform for final predictions
15        y_test_original = scalers_y[target].inverse_transform(y_test_scaled.
16                                                                reshape(-1, 1))
17        y_pred_test_original = scalers_y[target].inverse_transform(
18                                                                y_pred_test_scaled.reshape(-1, 1))
```

```

17
18     mse_final = mean_squared_error(y_test_original, y_pred_test_original)
19     mae_final = mean_absolute_error(y_test_original, y_pred_test_original)
20     r2_final = r2_score(y_test_original, y_pred_test_original)
21
22     print(f"\nFinal Test Results for {target} with Best Model:")
23     print(f"MSE: {mse_final:.4f}, MAE: {mae_final:.4f}, R2: {r2_final:.4f}")
24
25     # Plot Actual vs. Predicted
26     plt.figure(figsize=(10, 6))
27     plt.scatter(y_test_original, y_pred_test_original, alpha=0.6)
28     plt.plot([y_test_original.min(), y_test_original.max()],
29              [y_test_original.min(), y_test_original.max()], 'r--', label='
30              Perfect Prediction')
31
32     plt.title(f"Actual vs. Predicted for {target}")
33     plt.xlabel("Actual Values")
34     plt.ylabel("Predicted Values")
35     plt.legend()
36     plt.show()

```

Output plot:

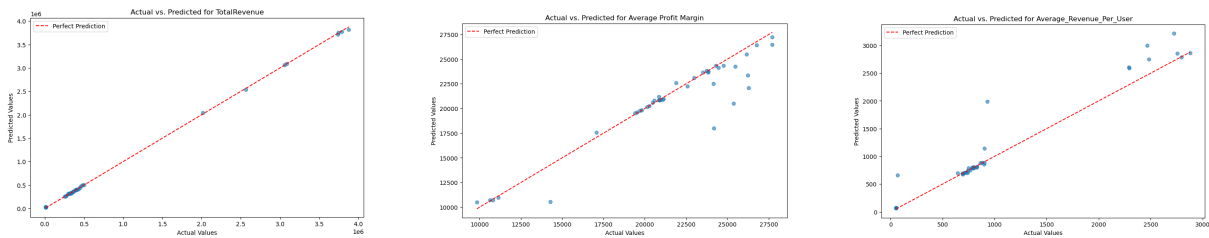


Figure 47: Actual vs Predicted

7.5.5 Future Predictions (Daily and Weekly)

In this section, future predictions are made for two different time periods: daily and weekly:

1. **Daily Predictions:** The models predict future values for the next day based on dynamically updated features. The feature update logic applies realistic trends such as slight increases in order quantity and minor fluctuations in price, discount, tax, and freight.
2. **Weekly Predictions:** The models also predict future values for the next week, with predictions made at weekly intervals. The same feature update logic is applied to simulate realistic changes in sales patterns over the weekly horizon.

The code for Weekly Predictions is quite similar to Daily Predictions so I only show Daily one.

```

1 # Extend the index for prediction (e.g., next day)
2 future_dates = pd.date_range(start=df.index[-1], periods=2, freq='D')[1:]
3

```

```

4 # Prepare initial raw features from the last known features
5 last_known_features = pd.DataFrame(
6     scaler_X.inverse_transform(pd.DataFrame(X_scaled[-1].reshape(1, -1), columns
7         =scaler_X.feature_names_in_)),
8     columns=scaler_X.feature_names_in_
9 )
10 # Initialize raw features for updates
11 X_future_raw = last_known_features.copy()
12
13 # Initialize scaled features and predictions
14 X_future_scaled = [X_scaled[-1]] # Start with the last known scaled feature set
15 future_predictions = {target: [] for target in targets}
16
17 # Define the feature update logic
18 def update_features(last_features):
19     """
20     Update features with realistic business logic and trends.
21     Args:
22         last_features (pd.Series): Previous day's features
23     Returns:
24         pd.Series: Updated features
25     """
26     updated_features = last_features.copy()
27
28     # Example business logic for trends or variability:
29     updated_features['OrderQty'] *= (1 + np.random.normal(0.01, 0.005)) # Trend
30                                     up
31     updated_features['UnitPrice'] *= (1 + np.random.normal(0, 0.002)) #
32                                     Stable
33     updated_features['UnitPriceDiscount'] = max(0, updated_features['
34         UnitPriceDiscount'] +
35                                     np.random.normal(0, 0.02)) #
36                                     Small fluctuation
37
38     updated_features['TaxAmt'] = updated_features['UnitPrice'] * 0.1 #
39                                     Proportional to UnitPrice
40     updated_features['Freight'] *= (1 + np.random.normal(0, 0.003)) #
41                                     Minor variations
42
43     return updated_features
44
45 # Predict future values dynamically for each target
46 for target in targets:
47     # Retrieve the best model for this target
48     model = None
49     for key, saved_model in best_models.items():
50         if key.startswith(target): # Match the target in the key
51             model = saved_model
52             break

```

```

46
47     if not model:
48         raise ValueError(f"No best model found for target: {target}")
49
50     # Loop through each future date
51     for _ in range(len(future_dates)):
52         # Predict using the current scaled features
53         current_prediction = model.predict(X_future_scaled[-1].reshape(1, -1))
54         future_predictions[target].append(current_prediction[0])
55
56         # Update raw features based on the last known raw features
57         next_raw_features = update_features(X_future_raw.iloc[-1])
58
59         # Append the updated raw features
60         X_future_raw = pd.concat(
61             [X_future_raw, pd.DataFrame([next_raw_features], columns=features)],
62             ignore_index=True
63         )
64
65         # Scale the updated features for model input
66         next_scaled_features = scaler_X.transform(next_raw_features.values.
67             reshape(1, -1))[0]
68         X_future_scaled.append(next_scaled_features)
69
70 # Convert predictions back to the original scale
71 future_predictions = {
72     target: scalers_y[target].inverse_transform(
73         np.array(future_predictions[target]).reshape(-1, 1)
74     ).flatten()
75     for target in targets
76 }
77
78 # Combine predictions into a DataFrame
79 future_predictions_df = pd.DataFrame(
80     future_predictions, index=future_dates, columns=targets
81 )
82
83 print("Future Predictions:")
84 print(future_predictions_df)

```

Output prediction for the next day:

Future Predictions:

	TotalRevenue	Average Profit Margin	Average_Revenue_Per_User
2014-07-01	6191.589636	14089.831903	1948.698455

Output prediction for the next week:

Future Predictions:

	TotalRevenue	Average Profit Margin	Average_Revenue_Per_User
2014-07-13	31767.932273	9929.660491	1565.579814

7.5.6 Load data to data warehouse

Since we need to show the efficiency of our model and display some relevant insights and data we would need to load some of our features and data of our EDA to the data warehouse. Here is the schema of the table:

Dim Data			
	Column Name	Data Type	Allow Nulls
	OrderDate	datetime	<input checked="" type="checkbox"/>
	ProductCategoryName	varchar(MAX)	<input checked="" type="checkbox"/>
	TerritoryID	bigint	<input checked="" type="checkbox"/>
	TerritoryName	varchar(MAX)	<input checked="" type="checkbox"/>
	SalesAmount	float	<input checked="" type="checkbox"/>
	[Average Profit Margin]	float	<input checked="" type="checkbox"/>
	OrderQty	float	<input checked="" type="checkbox"/>
	UnitPrice	float	<input checked="" type="checkbox"/>
	UnitPriceDiscount	float	<input checked="" type="checkbox"/>
	TaxAmt	float	<input checked="" type="checkbox"/>
	Freight	float	<input checked="" type="checkbox"/>
	Average_Revenue_Per_User	float	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

We also need to load the our regression result as well as the test set to evaluate the model so this is our dimension for that purpose:

Dim Prediction			
	Column Name	Data Type	Allow Nulls
	Date	datetime	<input checked="" type="checkbox"/>
	SalesAmount_Actual	float	<input checked="" type="checkbox"/>
	SalesAmount_Predicted	float	<input checked="" type="checkbox"/>
	[Average Profit Margin_Actual]	float	<input checked="" type="checkbox"/>
	[Average Profit Margin_Predicted]	float	<input checked="" type="checkbox"/>
	Average_Revenue_Per_User_Actual	float	<input checked="" type="checkbox"/>
	Average_Revenue_Per_User_Predicted	float	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

7.5.7 Notebook scheduling

Similar to the ETL pipeline we also need to schedule our notebook so that it can periodically save our data to the data warehouse

Notebook Jobs

Notebook Job Definitions

Notebook Job Definitions

Reload

Job definition name	Input filename	Created at	Schedule	Status	Actions
regression copy	regression copy.ipynb	12/16/2024, 9:47:58 PM	Every minute	Active	
extract_regression	extract_regression.ipynb	12/16/2024, 9:47:50 PM	Every minute	Active	
extract_metric_product.ipynb	extract_metric_product.ipynb	12/16/2024, 9:47:40 PM	Every minute	Active	

1-3 of 3 < >

There are 3 jobs that are currently active, scheduled to run every minute:

- `regression_copy.ipynb` is used to fit the model with data and load the data into `Dim_Prediction`
- `extract_regression.ipynb` is used for extract the features and data needed for regression and analysis
- `extract_metric_product.ipynb` is used for loading the features and data used for regression and analysis into the data warehouse

8 Use-case and recommendations

8.1 Addressing Low Performance in Germany and France

From our observation, Germany and France always have the lowest value among all Sales Territories in 3 sales metrics Total Revenue, Average Profit Margin, and Average Revenue Per User. This under-performance suggests that these markets face specific challenges compared to higher-performing regions.

Possible Reasons:

- Market Penetration Issues: limited brand awareness or presence in these territories may lead to lower customer engagement and sales.
- Weaker Customer Spending Power: economic conditions or consumer preferences in these regions may result in lower purchase quantities or preference for power-priced products.

Recommendations:

- Market-Specific Marketing Campaigns: Launch targeted marketing campaigns to increase brand awareness and customer engagement in Germany and France. Consider region-specific promotions and cultural preferences to better appeal to these markets.
- Enhance Customer Experience: Improve customer service, delivery speed, and return policies to enhance satisfaction and drive repeat business. Investing in localized support and service options may build trust with customers.
- Operational Optimization: Streamline logistics and distribution networks to reduce shipping costs and improve efficiency. Consider partnering with local distributors or fulfillment centers to reduce operational expenses.

8.2 Addressing Underperformance in Accessories

The Accessories category consistently underperforms in Total Revenue compared to other product categories, despite having the highest Profit Margin. This indicates that Accessories are a high-margin, low-volume product.

Possible Reasons:

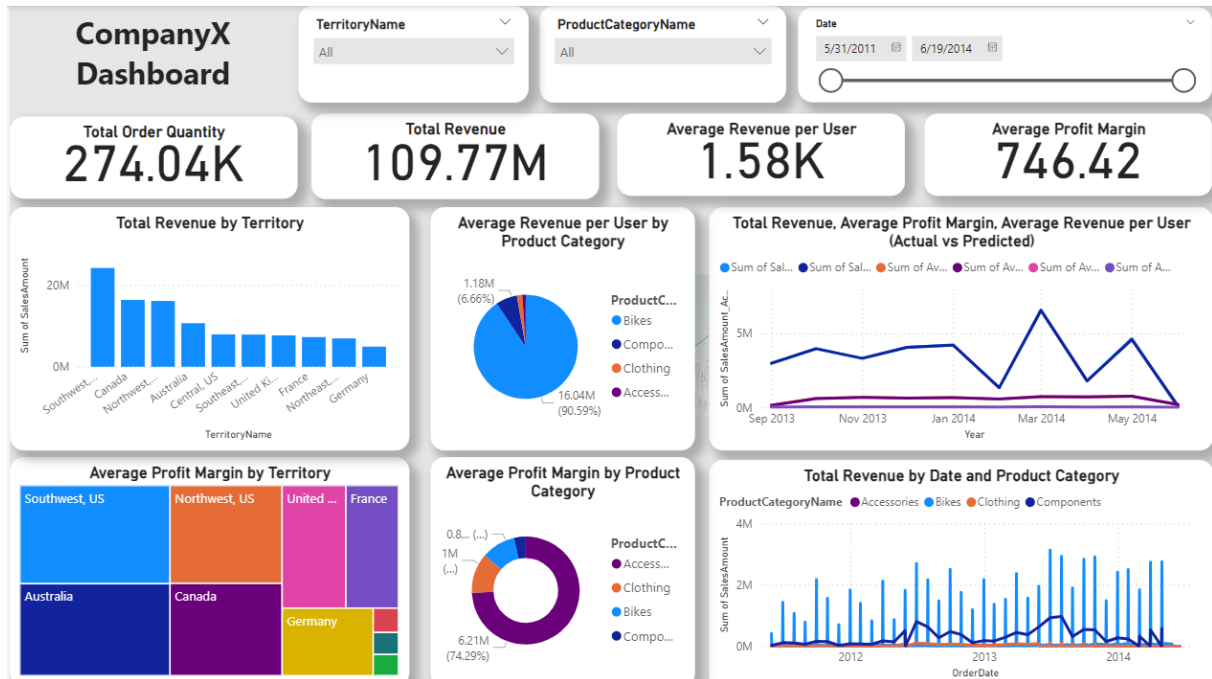
- Limited Market Demand: Accessories may cater to a niche market, resulting in lower sales volume.
- Lack of Promotion: Limited marketing efforts or visibility compared to high-revenue categories like Bikes may be hindering sales.
- Bundling Opportunities Not Leveraged: Accessories are often supplementary products, and a lack of bundling with primary items (e.g., Bikes) may be reducing sales opportunities.

Recommendations:

- Increase Marketing Efforts: Launch targeted campaigns to boost awareness and demand for Accessories, emphasizing their quality and utility.
- Bundling with High-Volume Products: Pair Accessories with Bikes or other popular products through bundled offers or discounts to increase sales volume.
- Incentivize Purchases: Introduce promotions, such as discounts or loyalty rewards, to encourage customers to add Accessories to their purchases.
- Product Placement Optimization: Highlight Accessories more prominently on sales platforms, websites, and stores to increase visibility and impulse purchases.

9 Power BI Dashboard

Here is our dashboard that contains some of the biggest highlight from our EDA:



The dashboard has indirect connection to the data warehouse where it always contains a snapshot of the data warehouse enabling user to access data immediately without internet connection, however it is inconvenient if users want to see live updates of the dashboard. In order for the dashboard to update user will have to refresh it. The dashboard features will include:

- Filter on top rows to filter out territory, country and timeline
- The dashboard also dynamically changes when user select a certain items in one charts
- There is also a line chart to check the accuracy of the prediction