ETL & ANALYTICS OF SUPERSTORE DATASET IN DIMENSIONAL DATA WAREHOUSE USING PYTHON

MET CS 779 Term Project Report

Abstract

My Term Project revolves around ETL of a Superstore dataset using Python. Creating a Dimensional Datawarehouse to answer some key Business questions and perform visualization of data using Tableau

Sushant Mohan Khot

sushantk@bu.edu

MET CS 779 Term Project Report

Table of Contents

1. Introduction

I am going to perform a ETL – Datawarehouse project on a Superstore dataset for a large size US store. The dataset consists of 4years of Order data (2015 to 2018). I am going to demonstrate ETL using Python. SQL Server is my choice of database to query the tables and get answers for my Business questions. I will also be utilizing Tableau for my query visualizations.

Additionally, the Constellation – Dimensional Datawarehouse will have FACT table and Dimension table with pre-aggregated and calculated fields in the Fact tables to help us answer some of our key Business questions.

- **Superstore Dataset:** It is a time series data of a Superstore transaction of Orders, Products, Customers etc. It is a retail dataset of a United States superstore for 4 Years.
- Any Business performing transactions on daily basis would like to analyze their data to understand their Customer behaviour, Popular products, Sales and many such entities to make better informed decisions to develop and grow their business.
- **Normalization of database:** I have demonstrated the Normalization technique on this dataset based on the skills acquired from this course. Although I have not uploaded any data in the normalized tables of the database, I have created the DDL to create the database structure and have designed a Normalized database ERD.
- **ETL into SQL SERVER using Python:** One of the key goals of the project is to perform ETL on this data using Python and loading the data into SQL SERVER. I have extracted a .csv format data of the superstore. This data needed cleansing, format changes and correction of inconsistent data which I have performed completely using Python.
- **Dimensional Datawarehouse:** I wanted to explore the topic of Dimensional Datawarehouse and implement a solution to answer some basic Business questions for the Superstore. The FACT tables and Dimensional Tables will help us answer some Key Business Questions.
- **Technologies used:** I have used Python for ETL, SQL SERVER as my Database to query for analytics and Tableau for visualizations.
-

2. Dataset

I found the Superstore dataset from Kaggle. It has 9800 rows. and 17 attributes. This dataset is a time series data of a Superstore transaction of Orders, Products, Customers etc. It is a retail dataset of a United States superstore for 4 Years.

The data can be categorized as shown below:

- **Order Data:** Order ID, Order Date, Ship Date, Ship Mode
- **Customer Data:** Customer ID, Customer Name, Segment
- **Location:** Country, City, State, Postal Code, Region
- **Product Data:** Product ID, Category, Sub-Category, Product Name

- **$ Sales:** Sales

Sample Dataset screenshot:

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category
1	CA-2017-152156	08-11-2017	11-11-2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South	FUR-BO-10001798	Furniture	Bookcases
2	CA-2017-152156	08-11-2017	11-11-2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420	South	FUR-CH-10000454	Furniture	Chairs
3	CA-2017-138688	12-06-2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036	West	OFF-LA-10000240	Office Supplies	Labels

Product Name	Sales
Bush Somerset Collection Bookcase	261.96
Hon Deluxe Fabric Upholstered Stacking Chairs, Rounded Back	731.94
Self-Adhesive Address Labels for Typewriters by Universal	14.62

I have also attached the dataset with my submission. The file name is "**SuperStore_dataset.csv**".

3. Business Questions

This dataset is for a large superstore which has many transactions. The transactions show us the Order details, Customer's details and the Product ordered along with the total sales for that product. It also has details like the Customer Segment, Product Categories and Subcategories.

I came up with the below Business Questions which I would like to get answers to quickly using my Dimensional Datawarehouse Design.

1. What are the TOP 3 most Popular Products in terms of Sales and at which Location?
2. Which Product Sub-category is the most popular?
3. Which Location has the maximum Average Sales and in which Month-Year?
4. Which are the TOP 5 Cities with Highest number of Orders placed?
5. Which Customer has the most Orders? What is their Segment?
6. Which Shipping Mode is most requested by the Customers?
7. What is the Maximum delay between Order and Ship Date? How many cities have this maximum delay?
8. Which Year had the most Sales?

4. Normalization – ERD

Below is the ERD for a Normalized design of the dataset. The ERD is color coded to categorize the tables in 4 sections.

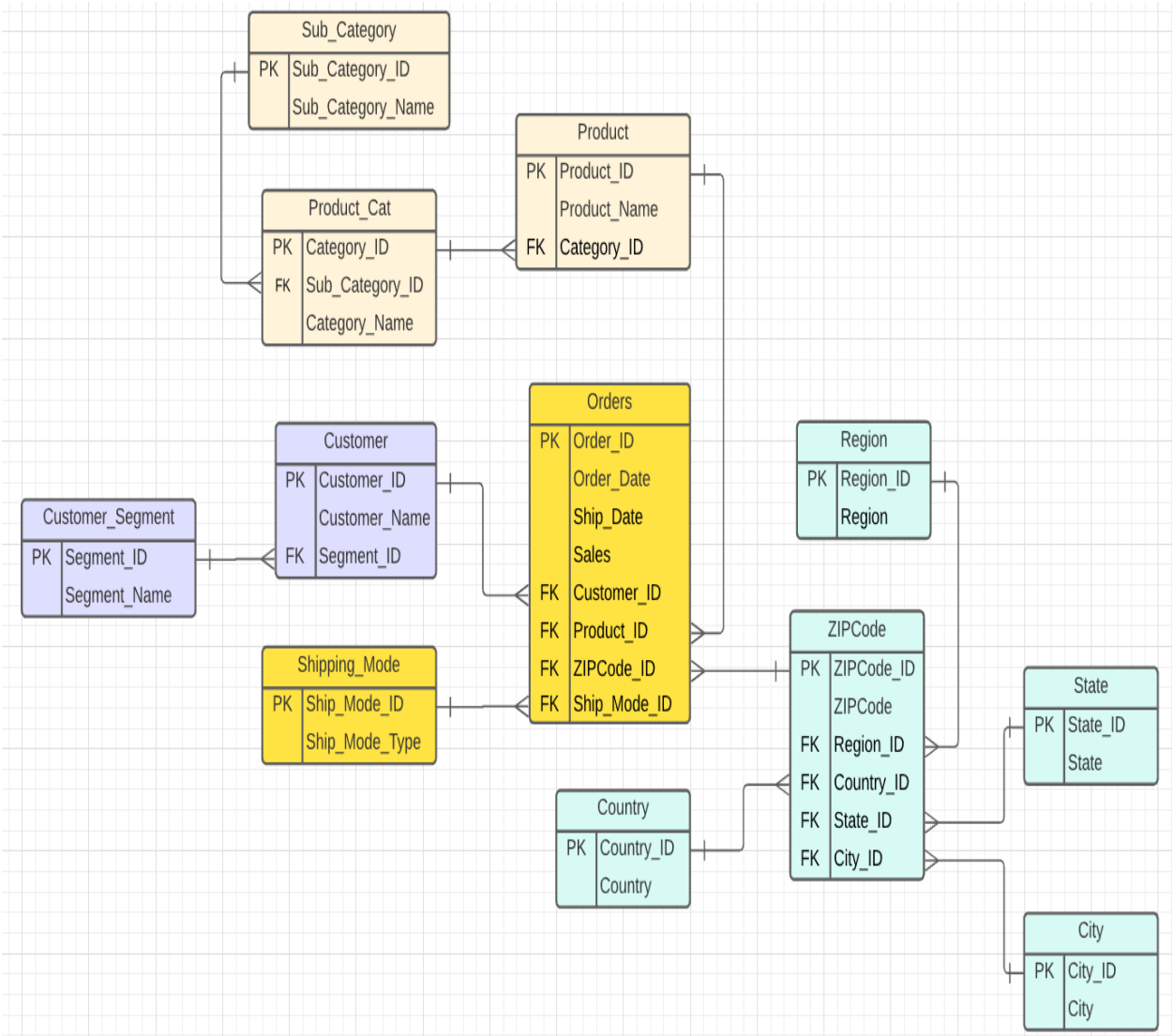
1. Product related tables: Product, Product_Category and Sub_Category

2. Customer related tables: Customer and Customer_Segment

3. Orders related Tables: Orders and Shipping_Mode

4. Location related tables: Region, Country, State, City and ZipCode

All these tables are normalized and share the foreign key constraints as shown in the ERD below.



5. ETL and Staging in SQL SERVER (Python and SQL SERVER)

I have created a "Superstore" database in SQL Server which will host the Datawarehouse. In order to perform the ETL process, I have created a Staging_Table which will store the raw data coming from the csv source.

The first step was to create the database and have the Staging table ready:

```
-- Create the Superstore Database
CREATE DATABASE Superstore
go

use Superstore;


-- STAGING Table for storing all our data in one Table to further distribute to different
FACT and Dimension Tables
CREATE TABLE Staging_Table(
Row_ID int IDENTITY(1,1),
Order_ID varchar(20),
Order_dt date,
Ship_dt date,
Ship_Mode varchar(20),
Customer_ID varchar(20),
Customer_Name varchar(100),
Segment varchar(20),
Country varchar(32),
City varchar(32),
State varchar(32),
Zip_Code varchar(10),
Region varchar(10),
Product_ID varchar(32),
Category varchar(32),
Subcategory varchar(32),
Product_Name varchar(255),
Sales numeric(8,2),
CONSTRAINT Superstore_RowId_PK PRIMARY KEY (Row_ID));
```

The next step will be to **EXTRACT** the data from .csv format using PYTHON. We are doing some **TRANSFORMATIONS**, data cleansing and handling null values in python using the **pandas dataframe**.

- We see that some of the date values have "-" and "/" as separators randomly. The date format in the csv is dd-mm-yyyy OR dd/mm/yyyy. We will make the format consistent by replacing "/" with "-".
- To handle the date format, we are going to use some Python code to format the dates so that Python can make the Order dates and Ship dates consistent across the rows.
- We can see that the Postal Code / ZIPCode column has blank values. We will handle this while inserting data in the Staging Table.
- We will replace all inconsistent Product names with one of the values from the list of assigned Product Names using Python code.

Python Code:

```
# Import Libraries
import os
import pyodbc
import pandas as pd
from datetime import datetime
import math


# Code to load the dataset using Relative Path
here = os.path.abspath(__file__)  # Relative Path code
input_dir = os.path.abspath(os.path.join(here, os.pardir))
superstore_dataset = os.path.join(input_dir, 'SuperStore_dataset.csv')


try:
    ss_df = pd.read_csv(superstore_dataset)

except Exception as e:
    print(e)
    print('failed to read Super Store data into Data Frame')


# COnnection to SQL Server
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=ARNAVDESKTOP;'
                      'Database=Superstore;'
                      'Trusted_Connection=yes;')


# Create a cursor for SQL code execution
cursor = conn.cursor()

# We can exclude the 1st "Row ID" column as it is set as an Identity column
in the Staging Table
ss_df = ss_df.iloc[:, 1:]

# Check if any columns in the dataframe have blank values
print(ss_df.isnull().any())

# We can see that the Postal Code / ZIPCode column has blank values.
# We will handle this while inserting data in the Staging Table.

# We see that the date values have "-" and "/" as separators randomly. The
date format in the csv is dd-mm-yyyy OR dd/mm/yyyy
# We will make the format consistent by replacing "/" with "-"
ss_df['Order Date'] = ss_df['Order Date'].str.replace('/','-')
ss_df['Ship Date'] = ss_df['Ship Date'].str.replace('/','-')

# Truncate the Staging Table in case we re-run this script multiple times on
the same csv to avoid duplication of records.
cursor.execute('TRUNCATE TABLE dbo.Staging_Table')


# Insert DataFrame records one by one into the Staging table.
for i,row in ss_df.iterrows():
```

```
    # Store the row values in a Python List
    val_list = list(row)

    # Convert the string values to Date
    val_list[1] = datetime.strptime(val_list[1], '%d-%m-%Y')
    val_list[2] = datetime.strptime(val_list[2], '%d-%m-%Y')

    # Check if ZIPCode value (val_list[10]) is blank then add a default
ZIPCode = 99999
    if math.isnan(val_list[10]):
        val_list[10] = 99999


    # Create the INSERT statement and execute it via the cursor connection
    # sql = "INSERT INTO dbo.Staging_ss ('" +cols + "') VALUES(?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
    sql = "INSERT INTO dbo.Staging_Table VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)"
    cursor.execute(sql, val_list)


# The below code will take care of inconsitent Product Names that I found in
the dataset.
# This will replace all inconsistent Product names with one of the values
from the list of assigned Product Names.
# For e.g. I saw that there were 32 Product IDs which had more than 1 Product
Name assigned.
cursor.execute('SELECT Product_ID FROM Staging_Table GROUP BY Product_ID
HAVING COUNT(DISTINCT(Product_Name)) > 1')
prod_ID = cursor.fetchall()

for prdID in prod_ID:
    cursor.execute("SELECT TOP 1 Product_Name FROM Staging_Table WHERE
Product_ID = '" + str(prdID[0]) + "'")
    prod_Name = cursor.fetchone()
    update_sql = "UPDATE Staging_Table SET Product_Name = ? WHERE Product_ID
= ?"
    cursor.execute(update_sql, [str(prod_Name[0]), str(prdID[0])])


# Commit and close the connection
conn.commit()
cursor.close()
conn.close()
```

6. Building the Datawarehouse (constellation)

Below is the ERD of the Datawarehouse that we are going to build to easily get answers to our Business questions:

We have 6 FACT tables tat have special fields created with calculated values based on our Business questions. All are Cumulative Fact tables.

We also have 6 Dimension tables. The DIM_PRODUCT is a Type 2: SCD New Record table while rest are Type 1: SCD Overwrite table

Fields from FACT tables used for answering our Business questions:

Notice the aggregated and calculated fields which we will load in our Datawarehouse tables.

1. **Total_Product_Sales_Loc** (FACT_Product_Sales_Cuml) will help us answer our 1st question of Top 3 most Popular Products and their Locations.

2. **Total_SubCategory_Sales_Year** (FACT_SubCategory_Sales_Cuml) will help us answer our 2nd Question on most popular Sub-Category with their Sales.

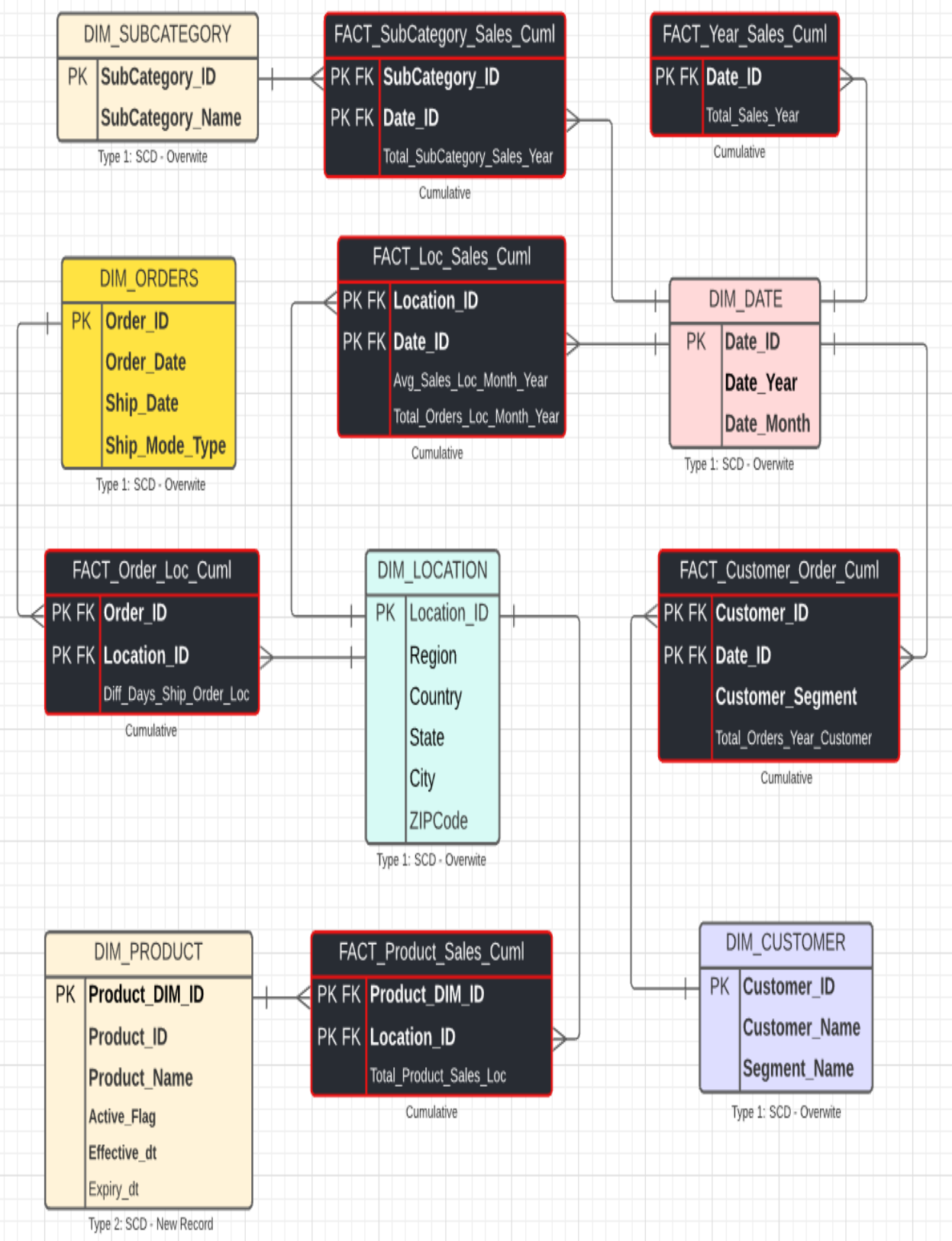
3. **Avg_Sales_Loc_Month_Year** and **Total_Orders_Loc_Month_Year** (FACT_Loc_Sales_Cuml) will help answer or 3rd question – Maximum Avg . Sales in Month-Year and 4th question – Top 5 Cities with highest numbers of Orders respectively.

4. **Total_Orders_Year_Customer** (FACT_Customer_Order_Cuml) will help us to answer our 5th question on Customers with most Orders and their Segments.

5. The most requested Shipping Mode can be a query on DIM_ORDERS table.

6. **Diff_Days_Ship_Order_Loc** (FACT_Order_Loc_Cuml) will help us get the Maximum difference in Days between Order Date and Ship date. Also, we can get the Number of Cities having this delay.

7. **Total_Sales_Year** (FACT_Year_Sales_Cuml) will help us get the Years from 2018 to 2015 in the order of Most Sales done in the respective years.


DIM_SUBCATEGORY
PK SubCategory_ID
SubCategory_Name
Type 1: SCD - Overwite
FACT_SubCategory_Sales_Cuml
PK FK SubCategory_ID
PK FK Date_ID
Total_SubCategory_Sales_Year
Cumulative
FACT_Year_Sales_Cuml
PK FK Date_ID
Total_Sales_Year
Cumulative
DIM_ORDERS
PK Order_ID
Order_Date
Ship_Date
Ship_Mode_Type
Type 1: SCD - Overwite
FACT_Loc_Sales_Cuml
PK FK Location_ID
PK FK Date_ID
Avg_Sales_Loc_Month_Year
Total_Orders_Loc_Month_Year
Cumulative
DIM_DATE
PK Date_ID
Date_Year
Date_Month
Type 1: SCD - Overwite
FACT_Order_Loc_Cuml
PK FK Order_ID
PK FK Location_ID
Diff_Days_Ship_Order_Loc
Cumulative
DIM_LOCATION
PK Location_ID
Region
Country
State
City
ZIPCode
Type 1: SCD - Overwite
FACT_Customer_Order_Cuml
PK FK Customer_ID
PK FK Date_ID
Customer_Segment
Total_Orders_Year_Customer
Cumulative
DIM_PRODUCT
PK Product_DIM_ID
Product_ID
Product_Name
Active_Flag
Effective_dt
Expiry_dt
Type 2: SCD - New Record
FACT_Product_Sales_Cuml
PK FK Product_DIM_ID
PK FK Location_ID
Total_Product_Sales_Loc
Cumulative
DIM_CUSTOMER
PK Customer_ID
Customer_Name
Segment_Name
Type 1: SCD - Overwite

7. Creating and Loading the Datawarehouse (Dimension and FACT tables)

We will now use SQL queries to calculate the special fields in the FACT tables and load the data from Staging into the FACT tables.

Creating the Datawarehouse Tables:

DIMENSION Tables:

```
/*
Dimension Tables
*/
-- Product Dimension
CREATE TABLE DIM_PRODUCT(
Product_DIM_ID int IDENTITY(1,1) NOT NULL,
Product_ID varchar(32) NOT NULL,
Product_Name varchar(255) NOT NULL,
Active_Flag varchar(1) DEFAULT 'Y' NOT NULL,
Effective_dt date DEFAULT GETDATE() NOT NULL,
Expiry_dt date,
CONSTRAINT DIM_Product_DIM_ID_PK PRIMARY KEY (Product_DIM_ID));

-- Orders Dimension
CREATE TABLE DIM_ORDERS(
Order_ID varchar(20) NOT NULL,
Order_Date date NOT NULL,
Ship_Date date NOT NULL,
Ship_Mode_Type varchar(20) NOT NULL,
CONSTRAINT DIM_Order_ID_PK PRIMARY KEY (Order_ID));

-- Customer Dimension
CREATE TABLE DIM_CUSTOMER(
Customer_ID varchar(20) NOT NULL,
Customer_Name varchar(100) NOT NULL,
Segment_Name varchar(20) NOT NULL,
CONSTRAINT DIM_Customer_ID_PK PRIMARY KEY (Customer_ID));

-- Date Dimension
CREATE TABLE DIM_DATE(
Date_ID date NOT NULL,
Date_Year numeric(4) NOT NULL,
Date_Month numeric(2) NOT NULL,
CONSTRAINT DIM_Date_ID_PK PRIMARY KEY (Date_ID));

-- Location Dimension
CREATE TABLE DIM_LOCATION(
Location_ID int IDENTITY(1,1) NOT NULL,
Region varchar(10) NOT NULL,
Country varchar(32) NOT NULL,
State varchar(32) NOT NULL,
City varchar(32) NOT NULL,
ZIPCode varchar(10) NOT NULL,
CONSTRAINT DIM_Location_ID_PK PRIMARY KEY (Location_ID));
```

```
-- Sub_Category Dimension
CREATE TABLE DIM_SUBCATEGORY(
SubCategory_ID int IDENTITY(1,1),
SubCategory_Name varchar(32) NOT NULL,
CONSTRAINT DIM_SubCategory_ID_PK PRIMARY KEY (SubCategory_ID));
```

FACT Tables

```
/*
FACT Tables
*/

-- Product_Sales_Cumulative Fact table
CREATE TABLE FACT_Product_Sales_Cuml(
Product_DIM_ID int NOT NULL,
Location_ID int NOT NULL,
Total_Product_Sales_Loc numeric(8,2) NOT NULL,
CONSTRAINT FACT_Product_Sales_Cuml_Product_Loc_ID_PK PRIMARY KEY (Product_DIM_ID,
Location_ID));

-- Customer_Order_Cumulative Fact table
CREATE TABLE FACT_Customer_Order_Cuml(
Customer_ID varchar(20) NOT NULL,
Date_ID int NOT NULL,
Customer_Segment varchar(20) NOT NULL,
Total_Orders_Year_Customer int NOT NULL,
CONSTRAINT FACT_Customer_Order_Cuml_Customer_Date_ID_PK PRIMARY KEY (Customer_ID,
Date_ID));

-- Order_Location Cumulative Fact table
CREATE TABLE FACT_Order_Loc_Cuml(
Order_ID varchar(20) NOT NULL,
Location_ID int NOT NULL,
Diff_Days_Ship_Order_Loc int NOT NULL,
CONSTRAINT FACT_Order_Loc_Cuml_Order_Loc_ID_PK PRIMARY KEY (Order_ID, Location_ID));

-- Location_Sales_Cumulative Fact table
CREATE TABLE FACT_Loc_Sales_Cuml(
Location_ID int NOT NULL,
Date_ID varchar(10) NOT NULL,
Avg_Sales_Loc_Month_Year numeric(8,2) NOT NULL,
Total_Orders_Loc_Month_Year int NOT NULL,
CONSTRAINT FACT_Loc_Sales_Cuml_Loc_Date_ID_PK PRIMARY KEY (Location_ID, Date_ID));

-- Year_Sales_Cumulative Fact table
CREATE TABLE FACT_Year_Sales_Cuml(
Date_ID int NOT NULL,
Total_Sales_Year numeric(8,2) NOT NULL,
CONSTRAINT FACT_Year_Sales_Cuml_Date_ID_PK PRIMARY KEY (Date_ID));
```

```
-- SubCategory_Sales_Cumulative Fact table
CREATE TABLE FACT_Subcategory_Sales_Cuml(
SubCategory_ID int NOT NULL,
Date_ID int NOT NULL,
Total_SubCategory_Sales_Year numeric(8,2) NOT NULL,
CONSTRAINT FACT_Subcategory_Sales_Cuml_SubCategory_Date_ID_PK PRIMARY KEY
(SubCategory_ID, Date_ID));
```

Loading the Datawarehouse:

Here we will be loading data into our Dimension and FACT tables. Notice the calculations done for the special fields which will be used for our Business analysis and Questions.

```
-- Correct the "San Diego" City ZipCode
UPDATE Staging_Table
SET City = 'San Diego'
WHERE Zip_Code = '92024'
```

Load data in DIMENSION tables:

```
/*
===================================
MERGE Data into the Dimension Tables
===================================
*/

-- PRODUCT Dimension FROM Staging
MERGE INTO DIM_PRODUCT AS tgt
USING (SELECT DISTINCT(Product_ID), Product_Name FROM  Staging_Table) AS src
ON src.Product_ID = tgt.Product_ID
WHEN NOT MATCHED BY TARGET THEN
       INSERT (Product_ID, Product_Name)
       VALUES (src.Product_ID, src.Product_Name)
WHEN MATCHED THEN UPDATE SET
       tgt.Product_ID = src.Product_ID,
       tgt.Product_Name = src.Product_Name;

--DELETE FROM DIM_PRODUCT
-- SELECT * FROM DIM_PRODUCT


-- CUSTOMER Dimension FROM Staging
MERGE INTO DIM_CUSTOMER AS tgt
USING (SELECT DISTINCT(Customer_ID), Customer_Name, Segment FROM  Staging_Table) AS src
ON src.Customer_ID = tgt.Customer_ID
WHEN NOT MATCHED BY TARGET THEN
       INSERT (Customer_ID, Customer_Name, Segment_Name)
       VALUES (src.Customer_ID, src.Customer_Name, src.Segment)
WHEN MATCHED THEN UPDATE SET
       tgt.Customer_ID = src.Customer_ID,
       tgt.Customer_Name = src.Customer_Name,
       tgt.Segment_Name = src.Segment;

--DELETE FROM DIM_CUSTOMER
-- Select * From DIM_CUSTOMER
```

```
-- SUBCATEGORY Dimension FROM Staging
MERGE INTO DIM_SUBCATEGORY AS tgt
USING (SELECT DISTINCT(Subcategory) FROM  Staging_Table) AS src
ON src.Subcategory = tgt.SubCategory_Name
WHEN NOT MATCHED BY TARGET THEN
	INSERT (SubCategory_Name)
	VALUES (src.Subcategory)
WHEN MATCHED THEN UPDATE SET
	tgt.SubCategory_Name = src.Subcategory;

--DELETE FROM DIM_SUBCATEGORY
-- Select * From DIM_SUBCATEGORY


-- LOCATION Dimension FROM Staging
MERGE INTO DIM_LOCATION AS tgt
USING (SELECT DISTINCT(Zip_Code), City, State, Country, Region FROM  Staging_Table) AS
src
ON src.Zip_Code = tgt.ZIPCode
WHEN NOT MATCHED BY TARGET THEN
	INSERT (ZIPCode, City, State, Country, Region)
	VALUES (src.Zip_Code, src.City, src.State, src.Country, src.Region)
WHEN MATCHED THEN UPDATE SET
	tgt.ZIPCode = src.Zip_Code,
	tgt.City = src.City,
	tgt.State = src.State,
	tgt.Country = src.Country,
	tgt.Region = src.Region;

--DELETE FROM DIM_LOCATION
-- Select * From DIM_LOCATION


-- ORDERS Dimension FROM Staging
MERGE INTO DIM_ORDERS AS tgt
USING (SELECT DISTINCT(Order_ID), Order_dt, Ship_dt, Ship_Mode FROM  Staging_Table) AS
src
ON src.Order_ID = tgt.Order_ID
WHEN NOT MATCHED BY TARGET THEN
	INSERT (Order_ID, Order_Date, Ship_Date, Ship_Mode_Type)
	VALUES (src.Order_ID, src.Order_dt, src.Ship_dt, src.Ship_Mode)
WHEN MATCHED THEN UPDATE SET
	tgt.Order_ID = src.Order_ID,
	tgt.Order_Date = src.Order_dt,
	tgt.Ship_Date = src.Ship_dt,
	tgt.Ship_Mode_Type = src.Ship_Mode;

--DELETE FROM DIM_ORDERS
-- Select * From DIM_ORDERS
```

```
-- DATE Dimension FROM Staging
MERGE INTO DIM_DATE AS tgt
USING (SELECT Order_dt AS Date_ID, DATEPART(YEAR, Order_dt) AS Date_Year, DATEPART(MONTH,
Order_dt) AS Date_Month FROM Staging_Table
          UNION
          SELECT Ship_dt AS Date_ID, DATEPART(YEAR, Ship_dt) AS Date_Year,
DATEPART(MONTH, Ship_dt) AS Date_Month FROM Staging_Table) AS src
ON src.Date_ID = tgt.Date_ID
WHEN NOT MATCHED BY TARGET THEN
       INSERT (Date_ID, Date_Year, Date_Month)
       VALUES (src.Date_ID, src.Date_Year, src.Date_Month)
WHEN MATCHED THEN UPDATE SET
       tgt.Date_ID = src.Date_ID,
       tgt.Date_Year = src.Date_Year,
       tgt.Date_Month = src.Date_Month;

--DELETE FROM DIM_DATE
-- Select * From DIM_DATE
```

Load data in FACT tables:

```
/*
================================
MERGE Data into the FACT Tables
================================
*/

-- Product_Sales FACT Table
MERGE INTO FACT_Product_Sales_Cuml AS tgt
USING (SELECT DP.PRODUCT_DIM_ID, DL.Location_ID, SUM(ST.Sales) AS Total_Product_Sales_Loc
          FROM Staging_Table ST
          JOIN DIM_PRODUCT DP
          ON ST.Product_ID = DP.Product_ID
          JOIN DIM_LOCATION DL
          ON ST.Zip_Code = DL.ZIPCode
          GROUP BY PRODUCT_DIM_ID, DL.Location_ID) AS src
ON (src.PRODUCT_DIM_ID = tgt.PRODUCT_DIM_ID AND
       src.Location_ID = tgt.Location_ID)
WHEN NOT MATCHED BY TARGET THEN
       INSERT (PRODUCT_DIM_ID, Location_ID, Total_Product_Sales_Loc)
       VALUES (src.PRODUCT_DIM_ID, src.Location_ID, src.Total_Product_Sales_Loc)
WHEN MATCHED THEN UPDATE SET
       tgt.PRODUCT_DIM_ID = src.PRODUCT_DIM_ID,
       tgt.Location_ID = src.Location_ID,
       tgt.Total_Product_Sales_Loc = src.Total_Product_Sales_Loc;

--DELETE FROM FACT_Product_Sales_Cuml
-- Select * from FACT_Product_Sales_Cuml


-- Order_Location FACT Table
MERGE INTO FACT_Order_Loc_Cuml AS tgt
USING (SELECT ST.Order_ID, DL.Location_ID, DATEDIFF(DAY, ST.Order_dt, ST.Ship_dt) AS
Diff_Days_Ship_Order_Loc
          FROM Staging_Table ST
          JOIN DIM_ORDERS DO
```

```
          ON ST.Order_ID = DO.Order_ID
          JOIN DIM_LOCATION DL
          ON ST.Zip_Code = DL.ZIPCode
          GROUP BY ST.Order_ID, DL.Location_ID, DATEDIFF(DAY, ST.Order_dt, ST.Ship_dt))
AS src
ON (src.Order_ID = tgt.Order_ID AND
       src.Location_ID = tgt.Location_ID)
WHEN NOT MATCHED BY TARGET THEN
       INSERT (Order_ID, Location_ID, Diff_Days_Ship_Order_Loc)
       VALUES (src.Order_ID, src.Location_ID, src.Diff_Days_Ship_Order_Loc)
WHEN MATCHED THEN UPDATE SET
       tgt.Order_ID = src.Order_ID,
       tgt.Location_ID = src.Location_ID,
       tgt.Diff_Days_Ship_Order_Loc = src.Diff_Days_Ship_Order_Loc;

--DELETE FROM FACT_Order_Loc_Cuml
-- Select * from FACT_Order_Loc_Cuml


-- Location_Sales FACT Table
MERGE INTO FACT_Loc_Sales_Cuml AS tgt
USING (SELECT DL.Location_ID, (CAST(DATEPART(MONTH, ST.Order_dt) AS VARCHAR) + '-' +
CAST(DATEPART(YEAR, ST.Order_dt) AS VARCHAR)) AS Date_ID, AVG(ST.Sales) AS
Avg_Sales_Loc_Month_Year, COUNT(DISTINCT(ST.Order_ID)) AS Total_Orders_Loc_Month_Year
          FROM Staging_Table ST
          JOIN DIM_LOCATION DL
          ON ST.Zip_Code = DL.ZIPCode
          JOIN DIM_DATE DD
          ON ST.Order_dt = DD.Date_ID
          GROUP BY DL.Location_ID, DATEPART(MONTH, ST.Order_dt), DATEPART(YEAR,
ST.Order_dt)) AS src
ON (src.Location_ID = tgt.Location_ID AND
       src.Date_ID = tgt.Date_ID)
WHEN NOT MATCHED BY TARGET THEN
       INSERT (Location_ID, Date_ID, Avg_Sales_Loc_Month_Year,
Total_Orders_Loc_Month_Year)
       VALUES (src.Location_ID, src.Date_ID, src.Avg_Sales_Loc_Month_Year,
src.Total_Orders_Loc_Month_Year)
WHEN MATCHED THEN UPDATE SET
       tgt.Location_ID = src.Location_ID,
       tgt.Date_ID = src.Date_ID,
       tgt.Avg_Sales_Loc_Month_Year = src.Avg_Sales_Loc_Month_Year,
       tgt.Total_Orders_Loc_Month_Year = src.Total_Orders_Loc_Month_Year;

--DELETE FROM FACT_Loc_Sales_Cuml
-- Select * from FACT_Loc_Sales_Cuml


-- Customer_Order FACT Table
MERGE INTO FACT_Customer_Order_Cuml AS tgt
USING (SELECT ST.Customer_ID, DATEPART(YEAR, DD.Date_ID) AS Date_ID, ST.Segment AS
Customer_Segment, COUNT(DISTINCT(ST.Order_ID)) AS Total_Orders_Year_Customer
          FROM Staging_Table ST
          JOIN DIM_CUSTOMER DC
          ON ST.Customer_ID = DC.Customer_ID
          JOIN DIM_DATE DD
          ON ST.Order_dt = DD.Date_ID
          GROUP BY ST.Customer_ID, ST.Segment, DATEPART(YEAR, DD.Date_ID)) AS src
```

```
ON (src.Customer_ID = tgt.Customer_ID AND
	src.Date_ID = tgt.Date_ID)
WHEN NOT MATCHED BY TARGET THEN
	INSERT (Customer_ID, Date_ID, Customer_Segment, Total_Orders_Year_Customer)
	VALUES (src.Customer_ID, src.Date_ID, src.Customer_Segment,
src.Total_Orders_Year_Customer)
WHEN MATCHED THEN UPDATE SET
	tgt.Customer_ID = src.Customer_ID,
	tgt.Date_ID = src.Date_ID,
	tgt.Customer_Segment = src.Customer_Segment,
	tgt.Total_Orders_Year_Customer = src.Total_Orders_Year_Customer;

--DELETE FROM FACT_Customer_Order_Cuml
-- Select * from FACT_Customer_Order_Cuml


-- Year_Sales FACT Table
MERGE INTO FACT_Year_Sales_Cuml AS tgt
USING (SELECT DATEPART(YEAR, ST.Order_dt) AS Date_ID, SUM(ST.Sales) AS Total_Sales_Year
	FROM Staging_Table ST
	JOIN DIM_DATE DD
	ON ST.Order_dt = DD.Date_ID
	GROUP BY DATEPART(YEAR, ST.Order_dt)) AS src
ON (src.Date_ID = tgt.Date_ID)
WHEN NOT MATCHED BY TARGET THEN
	INSERT (Date_ID, Total_Sales_Year)
	VALUES (src.Date_ID, src.Total_Sales_Year)
WHEN MATCHED THEN UPDATE SET
	tgt.Date_ID = src.Date_ID,
	tgt.Total_Sales_Year = src.Total_Sales_Year;

--DELETE FROM FACT_Year_Sales_Cuml
-- Select * from FACT_Year_Sales_Cuml


-- SubCategory_Sales FACT Table
MERGE INTO FACT_SubCategory_Sales_Cuml AS tgt
USING (SELECT DS.SubCategory_ID, DATEPART(YEAR, ST.Order_dt) AS Date_ID, SUM(ST.Sales) AS
Total_SubCategory_Sales_Year
	FROM Staging_Table ST
	JOIN DIM_SUBCATEGORY DS
	ON ST.Subcategory = DS.SubCategory_Name
	JOIN DIM_DATE DD
	ON ST.Order_dt = DD.Date_ID
	GROUP BY DS.SubCategory_ID, DATEPART(YEAR, ST.Order_dt)) AS src
ON (src.SubCategory_ID = tgt.SubCategory_ID AND
	src.Date_ID = tgt.Date_ID)
WHEN NOT MATCHED BY TARGET THEN
	INSERT (SubCategory_ID, Date_ID, Total_SubCategory_Sales_Year)
	VALUES (src.SubCategory_ID, src.Date_ID, src.Total_SubCategory_Sales_Year)
WHEN MATCHED THEN UPDATE SET
	tgt.SubCategory_ID = src.SubCategory_ID,
	tgt.Date_ID = src.Date_ID,
	tgt.Total_SubCategory_Sales_Year = src.Total_SubCategory_Sales_Year;


--DELETE FROM FACT_SubCategory_Sales_Cuml
-- Select * from FACT_SubCategory_Sales_Cuml
```

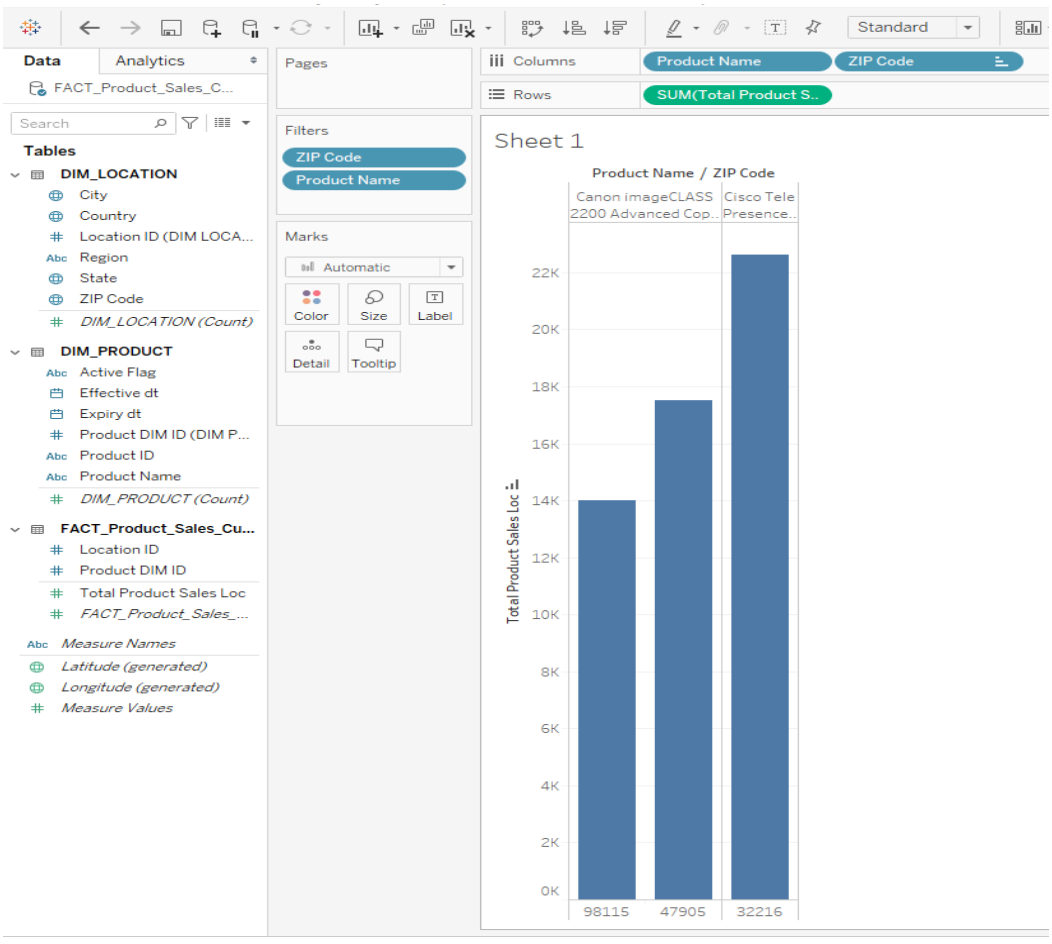
8. Answering Business Questions and Tableau Analytics

1. What are the TOP 3 most Popular Products in terms of Sales and at which Location?

```
SELECT TOP 3 DP.Product_ID,
                    DP.Product_Name,
                    DL.City,
                    DL.State,
                    DL.ZIPCode,
                    FP.Total_Product_Sales_Loc AS Total_Sales_$
FROM FACT_Product_Sales_Cuml FP
JOIN DIM_PRODUCT DP
ON FP.Product_DIM_ID = DP.Product_DIM_ID
JOIN DIM_LOCATION DL
ON FP.Location_ID = DL.Location_ID
ORDER BY FP.Total_Product_Sales_Loc desc
```

Results | Messages

	Product_ID	Product_Name	City	State	ZIPCode	Total_Sales_$
1	TEC-MA-10002412	Cisco TelePresence System EX90 Videoconferencing ...	Jacksonville	Florida	32216	22638.48
2	TEC-CO-10004722	Canon imageCLASS 2200 Advanced Copier	Lafayette	Indiana	47905	17499.95
3	TEC-CO-10004722	Canon imageCLASS 2200 Advanced Copier	Seattle	Washington	98115	13999.96

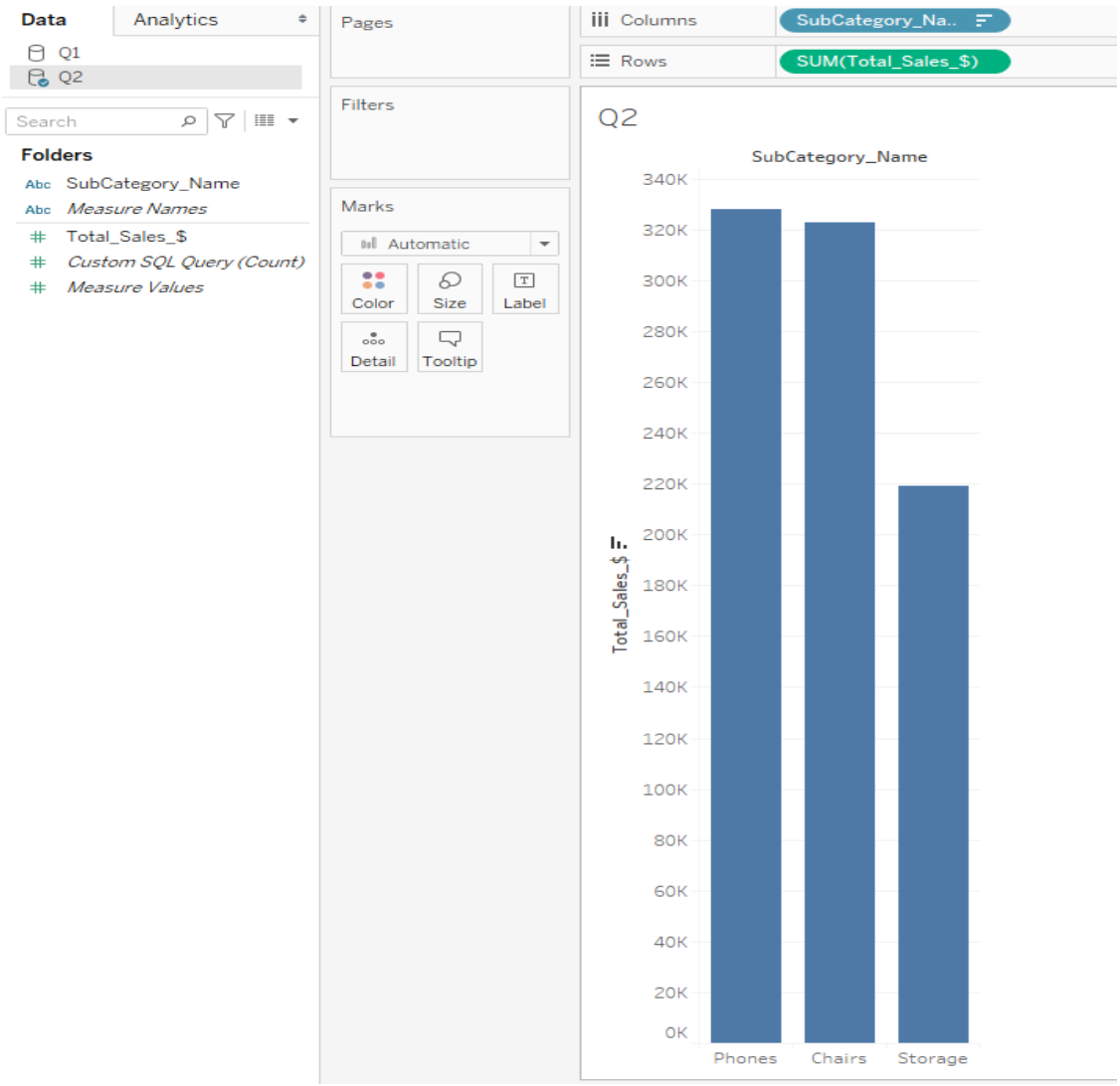


2. Which Product Sub-category is the most popular?

```
SELECT TOP 3 DS.SubCategory_Name,
          SUM(FS.Total_SubCategory_Sales_Year) AS Total_Sales_$
FROM FACT_Subcategory_Sales_Cuml FS
JOIN DIM_SUBCATEGORY DS
ON FS.SubCategory_ID = DS.SubCategory_ID
GROUP BY DS.SubCategory_Name
ORDER BY SUM(FS.Total_SubCategory_Sales_Year) desc
```

Results | Messages

	SubCategory_Name	Total_Sales_$
1	Phones	327782.49
2	Chairs	322822.71
3	Storage	219343.37

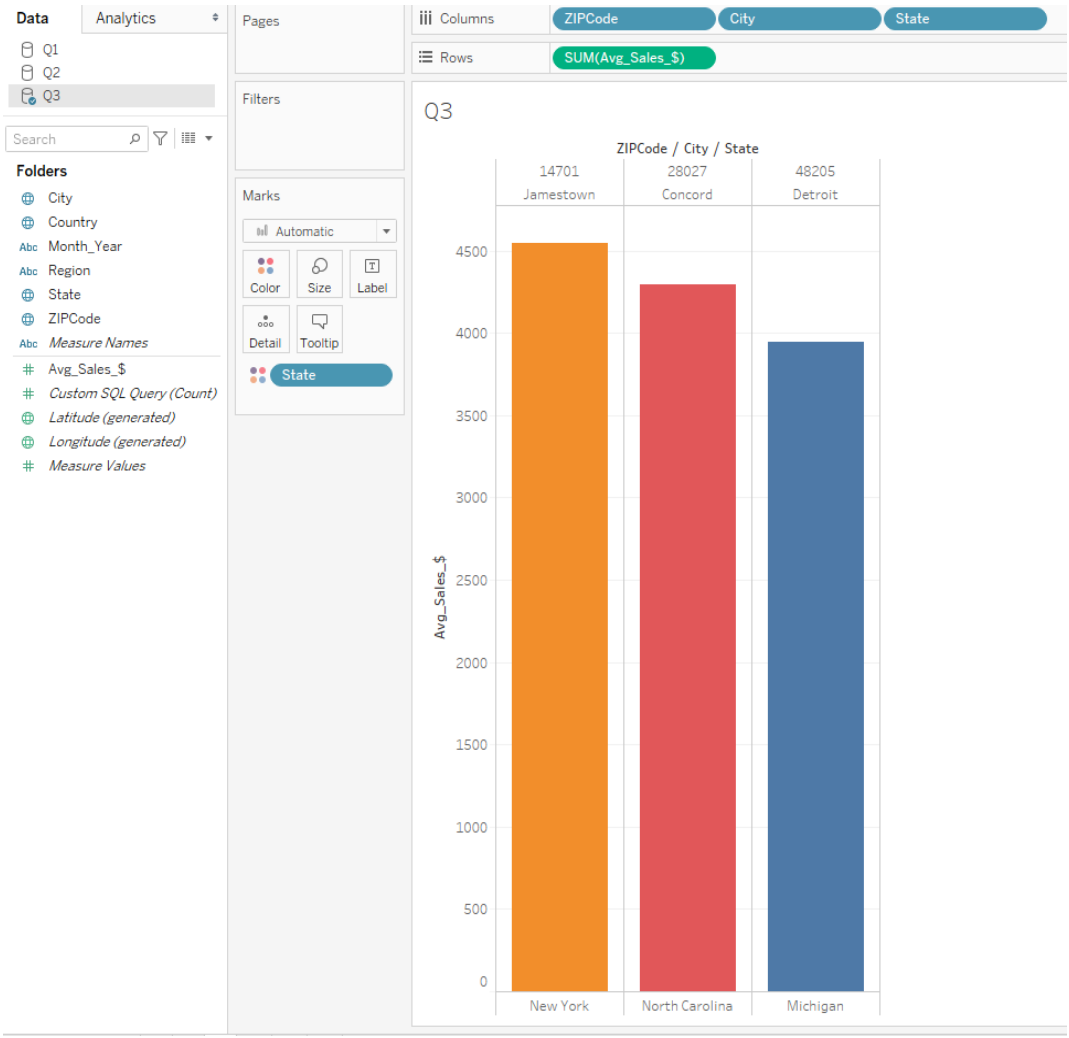


3. Which Location has the maximum Average Sales and in which Month-Year?

```
SELECT TOP 3 DL.City,
          DL.State,
          DL.Region,
          DL.Country,
          DL.ZIPCode,
          FL.Date_ID AS Month_Year,
          FL.Avg_Sales_Loc_Month_Year AS Avg_Sales_$
FROM FACT_Loc_Sales_Cuml FL
JOIN DIM_LOCATION DL
ON FL.Location_ID = DL.Location_ID
ORDER BY FL.Avg_Sales_Loc_Month_Year desc
```

Results Messages

	City	State	Region	Country	ZIPCode	Month_Year	Avg_Sales_$
1	Jamestown	New York	East	United States	14701	11-2015	4548.81
2	Concord	North Carolina	South	United States	28027	1-2016	4297.64
3	Detroit	Michigan	Central	United States	48205	12-2017	3947.35

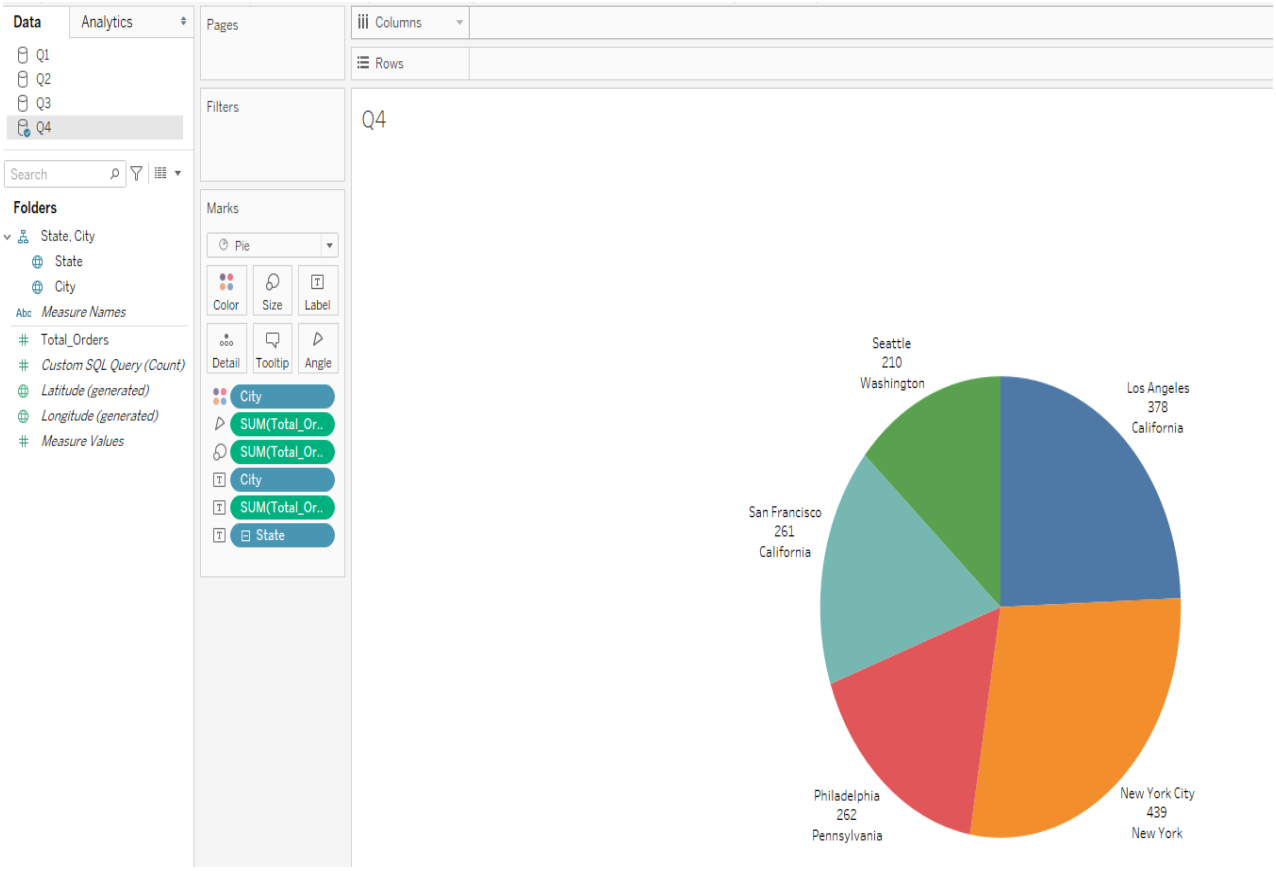


4. Which are the TOP 5 Cities with Highest number of Orders placed?

```
SELECT TOP 5 DL.City,
          DL.State,
          SUM(FL.Total_Orders_Loc_Month_Year) AS Total_Orders
FROM FACT_Loc_Sales_Cuml FL
JOIN DIM_LOCATION DL
ON FL.Location_ID = DL.Location_ID
GROUP BY DL.City,
               DL.State
ORDER BY SUM(FL.Total_Orders_Loc_Month_Year) desc
```

Results | Messages

	City	State	Total_Orders
1	New York City	New York	439
2	Los Angeles	California	378
3	Philadelphia	Pennsylvania	262
4	San Francisco	California	261
5	Seattle	Washington	210

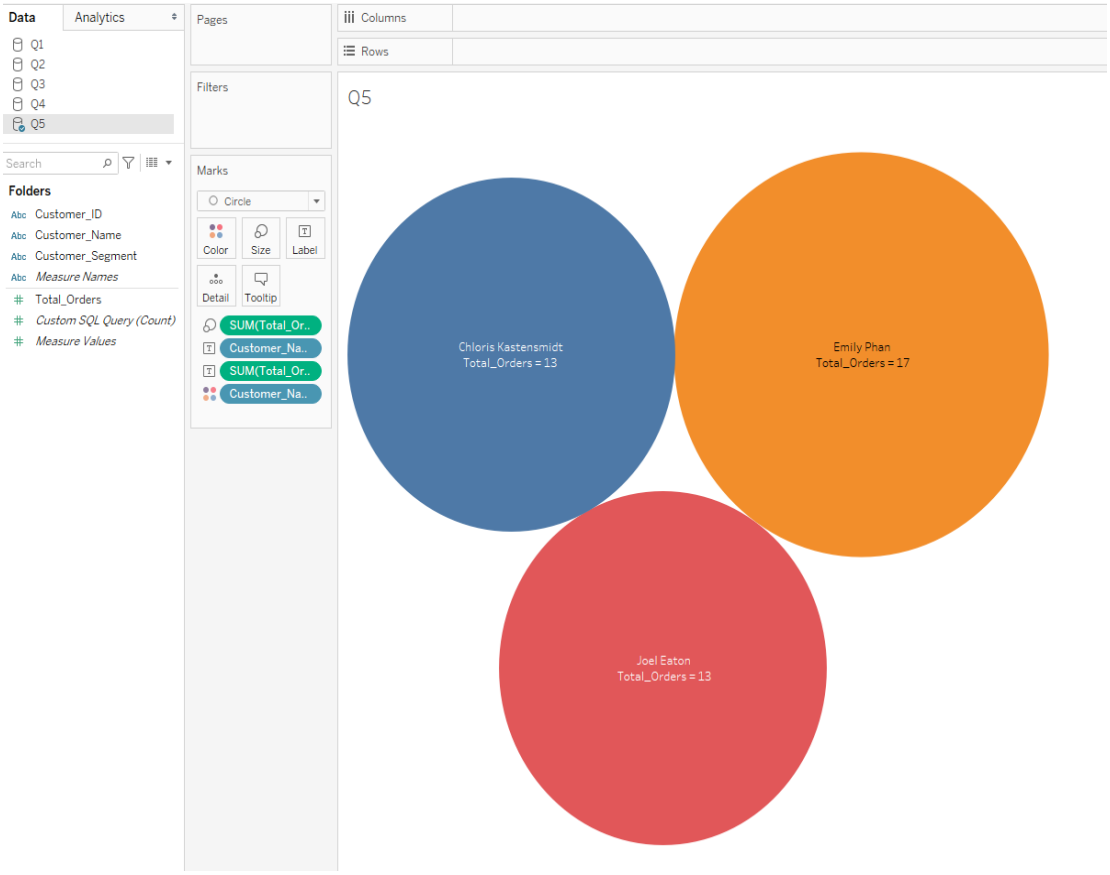


5. Which Customer has the most Orders? What is their Segment?

```
SELECT TOP 3 FC.Customer_ID,
          DC.Customer_Name,
          FC.Customer_Segment,
          SUM(FC.Total_Orders_Year_Customer) AS Total_Orders
FROM FACT_Customer_Order_Cuml FC
JOIN DIM_CUSTOMER DC
ON FC.Customer_ID = DC.Customer_ID
GROUP BY FC.Customer_ID,
          DC.Customer_Name,
          FC.Customer_Segment
ORDER BY SUM(FC.Total_Orders_Year_Customer) desc
```

Results Messages

	Customer_ID	Customer_Name	Customer_Segment	Total_Orders
1	EP-13915	Emily Phan	Consumer	17
2	CK-12205	Chloris Kastensmidt	Consumer	13
3	JE-15745	Joel Eaton	Consumer	13

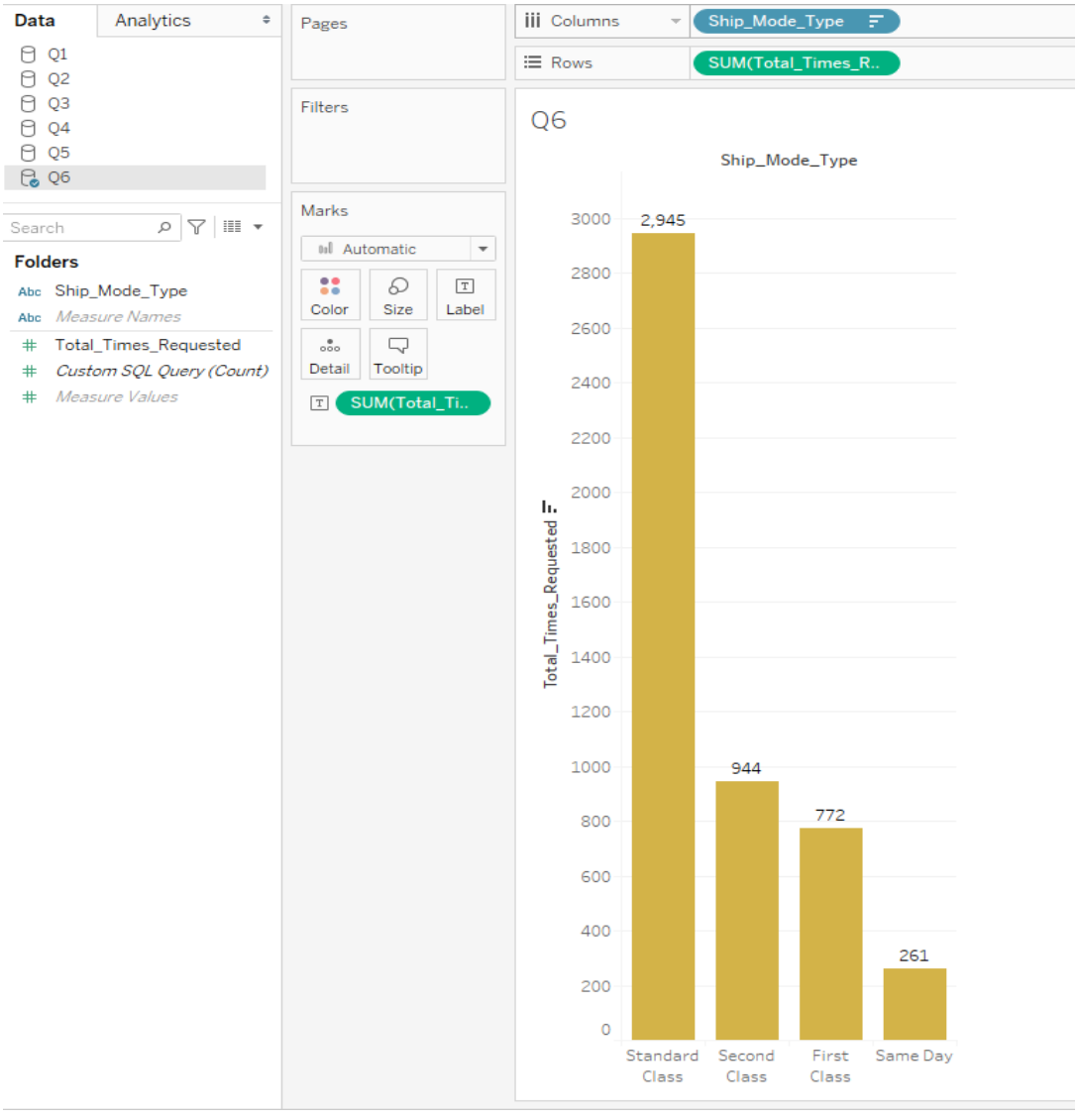


6. Which Shipping Mode is most requested by the Customers?

```
SELECT Ship_Mode_Type,
         COUNT(DISTINCT(Order_ID)) AS Total_Times_Requested
FROM DIM_ORDERS
GROUP BY Ship_Mode_Type
ORDER BY COUNT(DISTINCT(Order_ID)) desc
```

Results | Messages

	Ship_Mode_Type	Total_Times_Requested
1	Standard Class	2945
2	Second Class	944
3	First Class	772
4	Same Day	261



7. What is the Maximum delay between Order and Ship Date? How many cities have this maximum delay?

```
SELECT COUNT(DISTINCT(DL.City)) AS Num_of_Cities,
       MAX(FO.Diff_Days_Ship_Order_Loc) AS Maximum_Delay_in_Days
FROM FACT_Order_Loc_Cuml FO
JOIN DIM_LOCATION DL
ON FO.Location_ID = DL.Location_ID
ORDER BY MAX(FO.Diff_Days_Ship_Order_Loc) desc
```

Results Messages

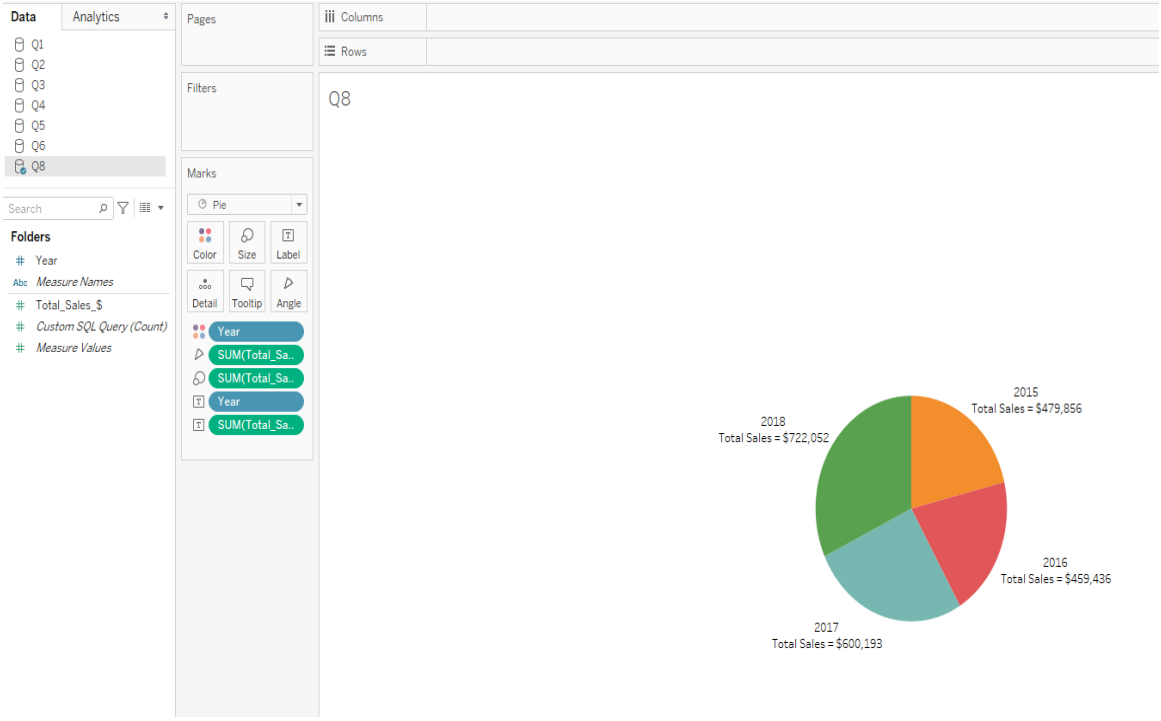
	Num_of_Cities	Maximum_Delay_in_Days
1	528	7

8. Which Year had the most Sales?

```
SELECT Date_ID AS Year,
       Total_Sales_Year AS Total_Sales_$
FROM FACT_Year_Sales_Cuml
ORDER BY Total_Sales_Year desc
```

Results Messages

	Year	Total_Sales_$
1	2018	722051.84
2	2017	600192.64
3	2015	479856.19
4	2016	459435.89



9. Conclusion

Thus, we conclude our ETL and Datawarehouse project by finding the answers for our Business questions. These will help the Business in the following manner:

- We will be able to easily find which are our Top Products that the Customers buy at different Locations. this will help us maintain appropriate pricing and stock of these products based on this Customer behavior.
- We will know the Subcategory of Products that are most popular amongst our customers, and we can concentrate on introducing more Products of these sub-categories to grow our business.
- We will also find out which specific Store Location is doing the most business and then we can make informed decisions on Inventory for this location, Scalability of the store and probably some discounts and perks for our customers at these specific locations.
- The same is applicable to the top cities who have the most Orders.
- Our analysis on Customers with the highest number of Orders can be used to introduce specific offers for these repeating customers and we can further analyze their transaction behavior, bundle offers based on their buying needs and their Segments etc.
- We also found out the most request Shipping Mode by our Customers. We can introduce offers like free shipping for our lowest mode of Shipping and try to introduce a subscription model for customers who opt for the most requested Shipping Mode.
- We can also look at the reason behind the delays between Order Dates and Ship dates for the specific cities. There could be issues with supply of materials, transportation or the overall supply chain might have to be optimized.
- Finally, we can analyze our data on Sales data for each Business year to find out what we did good on our most profitable year and what went wrong on our lowest profitable Year.

Bibliography

- How to connect Python to SQL Server using pyodbc. Retrieved from https://datatofish.com
- Superstore dataset. Retrieved from https://www.kaggle.com/