1.Introduction: Overview of Data Warehousing and Data Marts, and Their Significance in Business Intelligence

In the modern-day information-pushed enterprise environment, companies constantly are seeking methods to harness the good-sized quantities of information they generate and collect. The relevant answers for coping with big-scale analytical information are information warehouses and information marts. These structures are foundational to Business Intelligence (BI), supplying strong infrastructures for information storage, retrieval, and evaluation that empower agencies to extract actionable insights and assist strategic decision-making (Adewusi et al 2024).

An information warehouse is a centralized repository designed to aggregate, store, and control information from diverse heterogeneous assets throughout an organization. Its number one cause is to facilitate complicated analytical queries, long-time period information storage, and reporting capabilities. Unlike operational databases optimized for transaction processing, information warehouses are dependent for Online Analytical Processing (OLAP), which helps multidimensional queries and complicated calculations throughout big datasets (Li et al 2024).This information is generally historic and prepared the use of dimensional fashions together with big name schemas or snowflake schemas, which decorate question overall performance and simplify information retrieval for analysts and decision-makers (Nookala et al 2021).

Conversely, an information mart is a smaller, extra targeted model of an information warehouse. It is generally devoted to a selected enterprise area or branch together with marketing, finance, human resources, or sales (Gath et al 2024). While an information warehouse serves as a centralized hub for enterprise-extensive information evaluation, an information mart is customized to the specific desires of a specific consumer group.

Depending on their supply and structure, information marts may be dependent (sourced from a relevant information warehouse), independent (constructed without delay from operational structures), or hybrid (a mixture of both) (Nordeen et al 2020).Due to their narrower scope, information marts are less difficult and quicker to implement, making them perfect for departments that require brief get entry to applicable insights without the complexity of full-scale information warehousing.

The importance of information warehouses and information marts lies of their function in permitting Business Intelligence.BI entails equipment and practices that assist companies collect, integrate, analyse, and visualize enterprise records to assist higher selections. These structures offer a robust basis for forecasting, overall performance measurement, and strategic planning (Bussa et al 2023). With consolidated, cleaned, and historic information easily accessible, corporations can behaviour fashion analyses, come across anomalies, pick out patterns, and make information-knowledgeable selections that might now no longer be feasible the use of best operational structures. From a strategic point of view, the data warehouse supports comprehensive vision in organization by integrating data into all parts (Maswanganyi et al 2024).This visibility on the scale of the company is especially useful for executive managers, which must monitor various fields of activity areas to offer enlightenment strategic options (Loonam et al 2020).During this time, Data Marts provides specific information for the department, supports agility and decides quickly in individual trade units. For example, a financial department can use Mart data to budget or analyse service costs, while a marketing group can use them to monitor the effectiveness of the campaign.

2.Comparison and Business Justification: Data Warehouse vs Data Mart

Both a data mart and a data warehouse are essential parts of business intelligence systems. But when it comes to scale, scope, complexity, and use cases, they diverge greatly. Having a thorough grasp of these distinctions and practical examples aids in choosing the best option for an organization's analytical requirements.

1. Comparison of Data Warehouse and Data Mart

Feature	**Data Warehouse**	**Data Mart**
Scope	Enterprise-wide	Departmental or subject-specific
Data Volume	Very large (terabytes to petabytes	Smaller (gigabytes to terabytes
Source Systems	Integrates data from many departments	Often uses data from a few specific sources
Implementation Time	Integrates data from many departments	Often uses data from a few specific sources
Cost	Higher (infrastructure, storage, ETL)	Lower (simpler and smaller scale)
Users	Executives, analysts, enterprise-wide users	Specific departmental staff (e.g. Sales, HR)
Example	Amazon’s enterprise-wide data warehouse for global logistics and customer data analysis	Walmart’s sales department using data mart for weekly product sales performance

A **data warehouse** like that used by **Amazon** integrates data from logistics, marketing, customer service, and financial systems. This allows Amazon to analyze customer behavior globally, forecast inventory needs, and optimize delivery networks. It supports high-level business decisions by providing a comprehensive view across the company(Vidani et al 2024).

In contrast, a **data mart** example is found at **Walmart**, where individual departments like sales or inventory might use separate marts to analyze their own performance data. For instance, the marketing team could track promotional campaign results, while the inventory team uses a different data mart to monitor stock levels and product turnover(Akande et al 2021).

2. Business Justification

The size, structure, objectives, and data requirements of the business all influence the decision between a data warehouse and a data mart.

- Large Enterprises: A centralized data warehouse is more advantageous for businesses with several departments producing enormous amounts of data, such as banks, airlines, or e-commerce behemoths. It facilitates enterprise-wide reporting, guarantees consistency, and aids in the removal of data silos. For example, **Standard Chartered Bank** uses a data warehouse to consolidate data from its international branches for risk analysis, fraud detection, and regulatory compliance (Paleti et al 2025).
- **Small to Medium Enterprises (SMEs)**: According to Wang et at businesses with fewer departments or focused analytical requirements may find **data marts** more suitable. A **retail store chain** with a small team might only need a sales data mart to analyse customer purchases, trends, and seasonal demands (Boone et al 2019). This is cost-effective and quicker to implement than building a full-scale warehouse.
- **Agility vs Integration**: A **data mart** offers agility and simplicity. It allows departments to gain quick insights without depending on IT teams managing a large centralized system (Nookala et al 2022). However, for companies prioritizing **data governance, consistency, and long-term analytical capabilities**, a **data warehouse** provides greater strategic value.

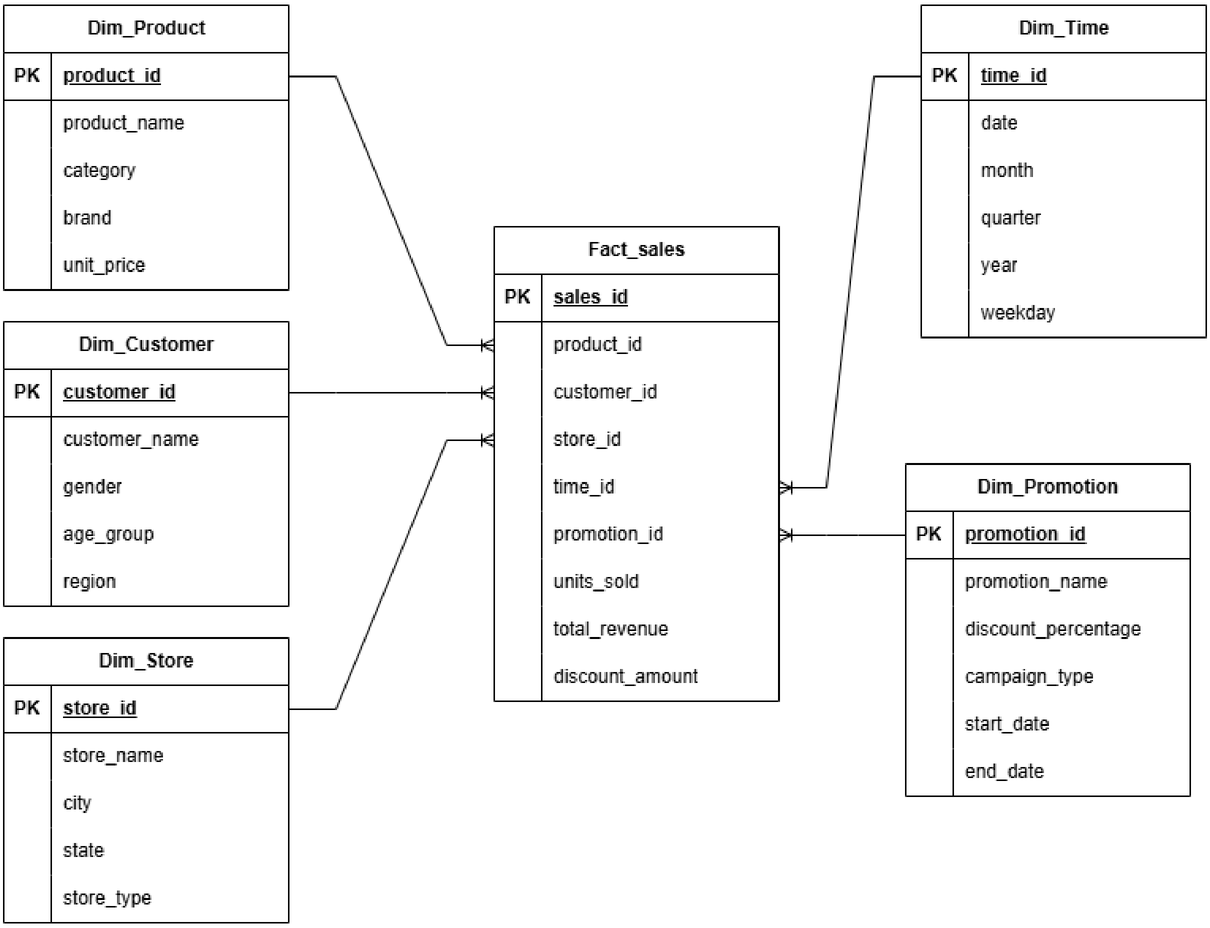
3.Star Schema Design for Sales and Marketing Business Scenario

1. Business Scenario

When considering a retail company that wants to analyse sales performance in many ways, such as stages, types of products, customer groups and geographical areas. The company must analyse significant sales parameters such as total income, selling units and depth to improve marketing efforts, rationalize inventory management and improve regional targeting tactics.

A star schema has been created to meet this analytical need. Product, Customer, Time, Store, and Promotion are some of the related dimension tables that make up the schema, along with the key fact table, Sales Fact, which records the quantitative sales data. These dimension tables allow for multidimensional analysis and provide the facts a descriptive context.

2. Star Schema Design



3. How the Star Schema Supports Business Analysis

The star schema is designed for analytical efficiency and simplicity. Here's how it enables insightful business analysis:

- **Sales Trend Analysis**: The business may measure sales performance by day, month, quarter, or year using time-series analysis using Dim_Time, detecting seasonal patterns or trends.
- **Product Performance**: Analysts may look at how various product categories or brands are doing in terms of units sold or money produced by combining Fact_Sales with Dim_Product.
- **Customer Segmentation**: By using Dim_Customer, the business may examine purchases according to consumer attributes like gender, age group, or area, enabling more individualized marketing tactics.
- **Geographical Insights**: Dim_Store enables geographic analysis, helping identify which stores or regions are underperforming or excelling.
- **Promotion Effectiveness**: By linking Dim_Promotion with sales data, the impact of marketing campaigns and discounts on overall sales can be evaluated, enabling better allocation of promotional budgets.

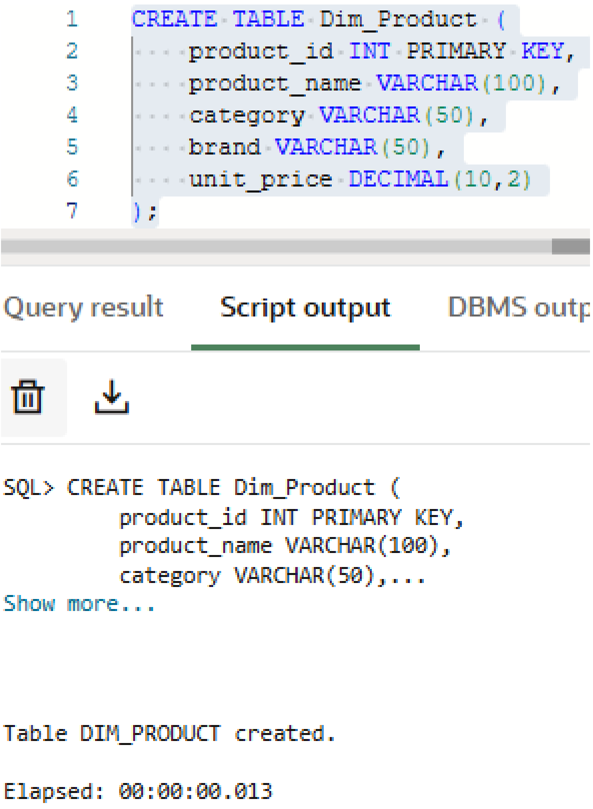
Star Schema's advantages for business users:

- **Simplified Queries**: Easier SQL queries are made possible by the denormalized structure, making them perfect for BI applications such as Power BI, Tableau, or Oracle BI.
- **High Performance**: Even with big datasets, indexing the foreign keys enables quick joins and query execution.
- **Scalability**: The schema can be expanded with additional dimensions or measures as the business grows

4. SQL Data Definitions: Star Schema CREATE Statements

The following SQL CREATE TABLE statements define the fact and dimension tables for the star schema with appropriate primary keys (PK), foreign keys (FK), and data types to maintain referential integrity.

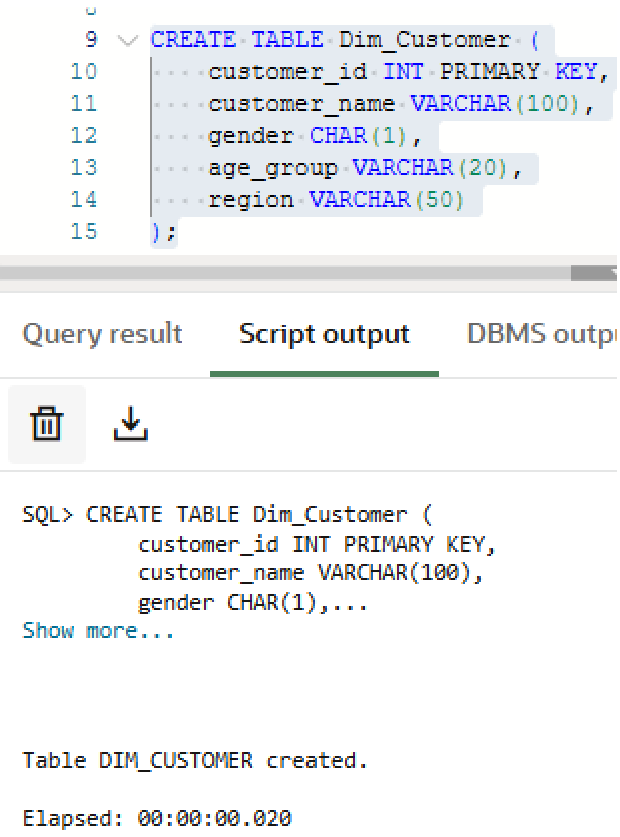
Creating dimension table for Product



```sql
CREATE TABLE Dim_Product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    brand VARCHAR(50),
    unit_price DECIMAL(10,2)
);
```

```
SQL> CREATE TABLE Dim_Product (
        product_id INT PRIMARY KEY,
        product_name VARCHAR(100),
        category VARCHAR(50),...
Show more...


Table DIM_PRODUCT created.

Elapsed: 00:00:00.013
```

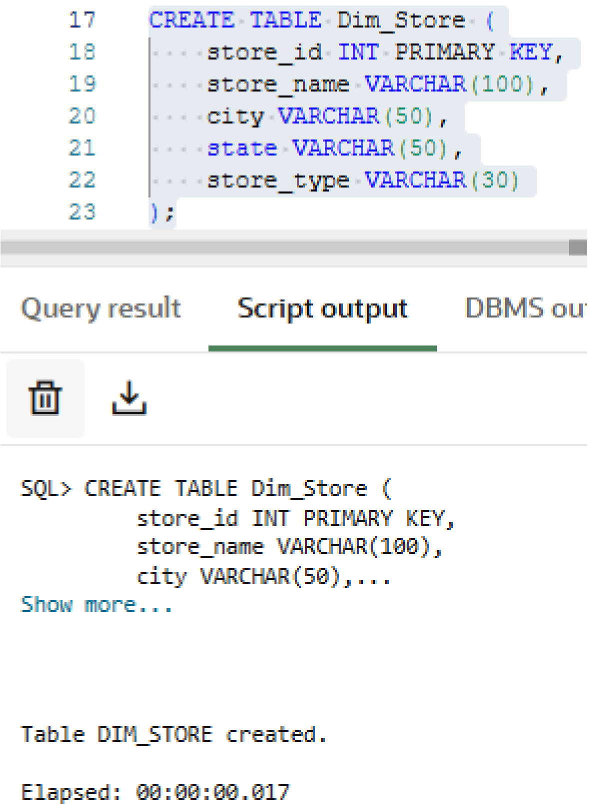
Creating dimension table for Customer



```sql
CREATE TABLE Dim_Customer (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    gender CHAR(1),
    age_group VARCHAR(20),
    region VARCHAR(50)
);
```

```
SQL> CREATE TABLE Dim_Customer (
         customer_id INT PRIMARY KEY,
         customer_name VARCHAR(100),
         gender CHAR(1),...
Show more...



Table DIM_CUSTOMER created.

Elapsed: 00:00:00.020
```

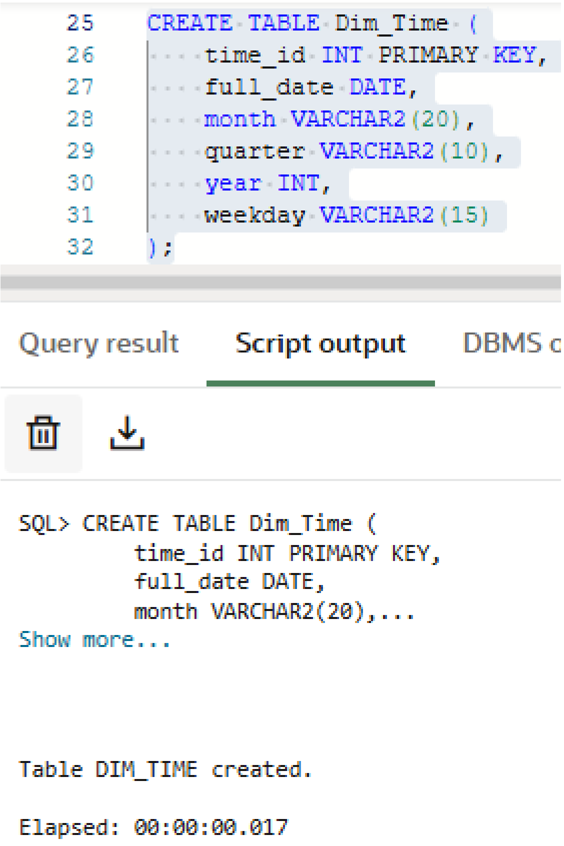
Creating dimension table for store



```sql
CREATE TABLE Dim_Store (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100),
    city VARCHAR(50),
    state VARCHAR(50),
    store_type VARCHAR(30)
);
```

```
SQL> CREATE TABLE Dim_Store (
         store_id INT PRIMARY KEY,
         store_name VARCHAR(100),
         city VARCHAR(50),...
Show more...



Table DIM_STORE created.

Elapsed: 00:00:00.017
```

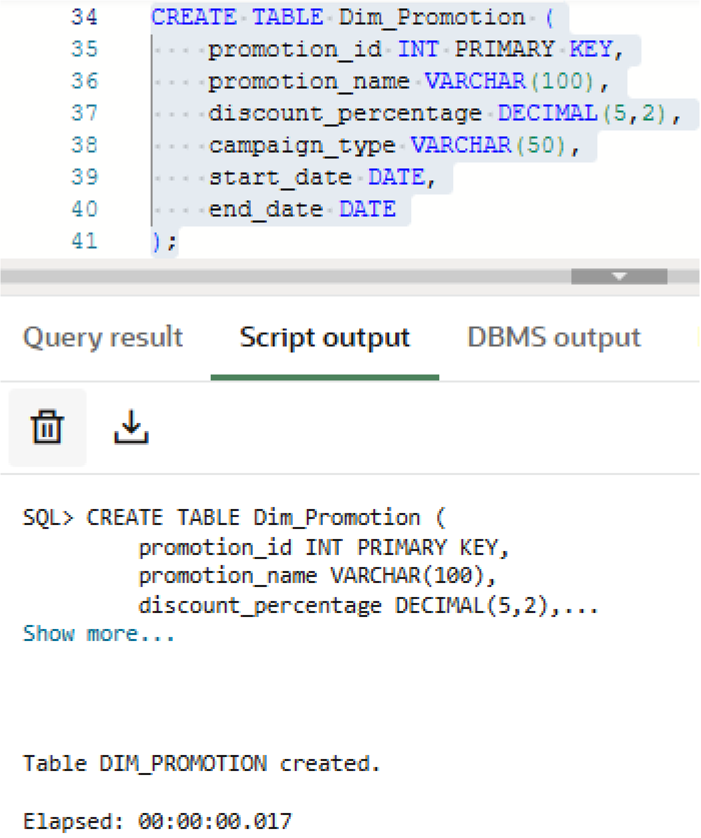
Creating dimension table for Time



```
CREATE TABLE Dim_Time (
    time_id INT PRIMARY KEY,
    full_date DATE,
    month VARCHAR2(20),
    quarter VARCHAR2(10),
    year INT,
    weekday VARCHAR2(15)
);
```

```
SQL> CREATE TABLE Dim_Time (
         time_id INT PRIMARY KEY,
         full_date DATE,
         month VARCHAR2(20),...
Show more...


Table DIM_TIME created.

Elapsed: 00:00:00.017
```

Creating



```
CREATE TABLE Dim_Promotion (
    promotion_id INT PRIMARY KEY,
    promotion_name VARCHAR(100),
    discount_percentage DECIMAL(5,2),
    campaign_type VARCHAR(50),
    start_date DATE,
    end_date DATE
);
```

```
SQL> CREATE TABLE Dim_Promotion (
         promotion_id INT PRIMARY KEY,
         promotion_name VARCHAR(100),
         discount_percentage DECIMAL(5,2),...
Show more...


Table DIM_PROMOTION created.

Elapsed: 00:00:00.017
```

```
CREATE TABLE Fact_Sales (
   sales_id INT PRIMARY KEY,
   product_id INT,
   customer_id INT,
   store_id INT,
   time_id INT,
   promotion_id INT,
   units_sold INT,
   total_revenue DECIMAL(12,2),
   discount_amount DECIMAL(10,2),

   FOREIGN KEY (product_id) REFERENCES Dim_Product(product_id),
   FOREIGN KEY (customer_id) REFERENCES Dim_Customer(customer_id),
   FOREIGN KEY (store_id) REFERENCES Dim_Store(store_id),
   FOREIGN KEY (time_id) REFERENCES Dim_Time(time_id),
   FOREIGN KEY (promotion_id) REFERENCES Dim_Promotion(promotion_id)
);
```

5. Business Intelligence Queries: Key Analytical Questions

A well-structured data warehouse empowers businesses to run insightful queries that support decision-making, performance monitoring, and strategic planning. Below are **eight (8)** key Business Intelligence (BI) queries that the star schema will support for a **retail company**:

1. Total Sales Revenue by Month and Year

Purpose: To monitor revenue trends over time and detect seasonality.
Query:

```
SELECT t.year, t.month, SUM(f.total_revenue) AS monthly_revenue
FROM Fact_Sales f
JOIN Dim_Time t ON f.time_id = t.time_id
GROUP BY t.year, t.month
ORDER BY t.year, t.month;
```

Insight: Helps management forecast future revenue and optimize marketing spend during peak periods.

2. Top 10 Best-Selling Products by Revenue

Purpose: To identify products that contribute most to revenue.
Query:

```
SELECT p.product_name, SUM(f.total_revenue) AS revenue

FROM Fact_Sales f

JOIN Dim_Product p ON f.product_id = p.product_id

GROUP BY p.product_name

ORDER BY revenue DESC

FETCH FIRST 10 ROWS ONLY;
```

Insight: Aids product prioritization, pricing strategy, and inventory planning.

3. Sales by Store and Region

Purpose: To compare performance across different store locations.
Query:

```
SELECT s.city, s.state, SUM(f.total_revenue) AS store_revenue

FROM Fact_Sales f

JOIN Dim_Store s ON f.store_id = s.store_id

GROUP BY s.city, s.state;
```

Insight: Helps in evaluating location-specific performance and making expansion or closure decisions.

4. Customer Purchase Behavior by Age Group

Purpose: To analyze how different customer demographics contribute to sales.
Query:

```
SELECT c.age_group, SUM(f.total_revenue) AS revenue

FROM Fact_Sales f

JOIN Dim_Customer c ON f.customer_id = c.customer_id

GROUP BY c.age_group;
```

Insight: Helps create customized marketing plans and targeted promotions.

5. Promotional Effectiveness

Purpose: To evaluate which advertising campaigns, stimulate sales higher
Query:

```
SELECT p.promotion_name, SUM(f.total_revenue) AS promo_revenue

FROM Fact_Sales f

JOIN Dim_Promotion p ON f.promotion_id = p.promotion_id

GROUP BY p.promotion_name;
```

Insight: Supports marketing optimization and budget allocation for future campaigns.

6. Units Sold vs. Discounts Given

Purpose: To assess the impact of discounts on product selling volume.
Query:

```
SELECT p.product_name, SUM(f.units_sold) AS units, SUM(f.discount_amount) AS total_discount

FROM Fact_Sales f

JOIN Dim_Product p ON f.product_id = p.product_id

GROUP BY p.product_name;
```

Insight: Determines whether discounts are increasing volume enough to justify margin loss.

7. Average Basket Size per Customer

Purpose: To comprehend how people make purchases.
Query:

```
SELECT c.customer_id, COUNT(f.sales_id) AS transactions, SUM(f.units_sold) AS total_units

FROM Fact_Sales f

JOIN Dim_Customer c ON f.customer_id = c.customer_id

GROUP BY c.customer_id;
```

Insight: Supports customer segmentation and loyalty program strategies.

8. Sales Performance by Day of the Week

Purpose: To find out which days generate the most sales.
Query:

```
SELECT t.weekday, SUM(f.total_revenue) AS weekday_sales

FROM Fact_Sales f

JOIN Dim_Time t ON f.time_id = t.time_id

GROUP BY t.weekday;
```

Reference list

Adewusi, A.O., Okoli, U.I., Adaga, E., Olorunsogo, T., Asuzu, O.F. and Daraojimba, D.O., 2024. Business intelligence in the era of big data: A review of analytical tools and competitive advantage. *Computer Science & IT Research Journal*, *5*(2), pp.415-431.

Li, X., Shen, Q. and Yang, T., 2024. Design and optimization of multidimensional data models for enhanced OLAP query performance and data analysis. *Applied and Computational Engineering*, *69*, pp.161-166.

Nookala, G., 2021. Evolution of Dimensional Modeling: Incorporating Big Data into Data Models. *Journal of Big Data and Smart Systems*, *2*(1).

Gath, S., 2024. *Principle of Data warehousing*. Academic Guru Publishing House.

Nordeen, A., 2020. *Learn Data Warehousing in 24 Hours*. Guru99.

Bussa, S., 2023. Enhancing BI tools for improved data visualization and insights. *International Journal of Computer Science and Mobile Computing*, *12*(2), pp.70-92.

Maswanganyi, N., Fumani, N., Khoza, J.K., Thango, B. and Lerato, M., 2024. Evaluating the impact of database and data warehouse technologies on organizational performance: A systematic review. *Available at SSRN 4997368*.

Loonam, J., Zwiegelaar, J., Kumar, V. and Booth, C., 2020. Cyber-resiliency for digital enterprises: a strategic leadership perspective. *IEEE Transactions on Engineering Management*, *69*(6), pp.3757-3770.

Vidani, J., 2024. E-Commerce Supply Chain Efficiency: A Case Study of Amazon E-Commerce Company. *Available at SSRN 4849852*.

Akande, Y.F., Idowu, J., Misra, A., Misra, S., Akande, O.N. and Ahuja, R., 2021, October. Application of xgboost algorithm for sales forecasting using walmart dataset. In *International Conference on Advances in Electrical and Computer Technologies* (pp. 147-159). Singapore: Springer Nature Singapore.

Paleti, S., 2025. *Smart Finance: Artificial Intelligence, Regulatory Compliance, and Data Engineering in the Transformation of Global Banking*. Deep Science Publishing.

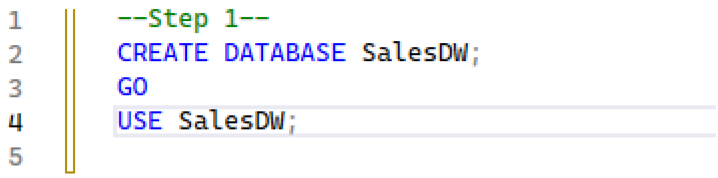
Boone, T., Ganeshan, R., Jain, A. and Sanders, N.R., 2019. Forecasting sales in the supply chain: Consumer analytics in the big data era. *International journal of forecasting*, *35*(1), pp.170-180.

Wang, S. and Wang, H., 2020. Big data for small and medium-sized enterprises (SME): a knowledge management model. *Journal of Knowledge Management*, *24*(4), pp.881-897.

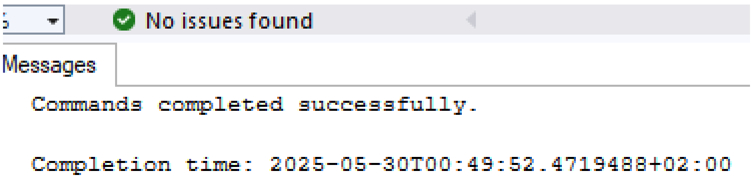
Nookala, G., 2022. Improving Business Intelligence through Agile Data Modeling: A Case Study. *Journal of Computational Innovation*, *2*(1).

TASK 2

STEP 1: CREATING A DATABASE IN SQL SERVER MANAGEMENT STUDIO



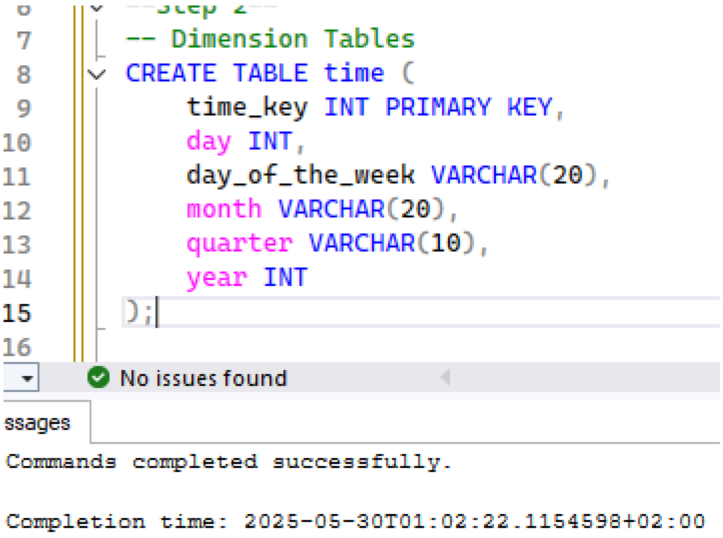
```
--Step 1--
CREATE DATABASE SalesDW;
GO
USE SalesDW;
```



```
Commands completed successfully.

Completion time: 2025-05-30T00:49:52.4719488+02:00
```

STEP 2: CREATING DIMENSION TABLE AND FACT TABLE

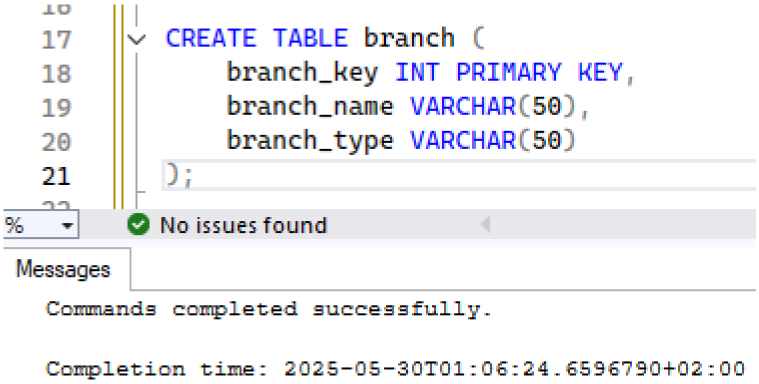
Time dimension



```
--Step 2--
-- Dimension Tables
CREATE TABLE time (
    time_key INT PRIMARY KEY,
    day INT,
    day_of_the_week VARCHAR(20),
    month VARCHAR(20),
    quarter VARCHAR(10),
    year INT
);
```

```
Commands completed successfully.

Completion time: 2025-05-30T01:02:22.1154598+02:00
```

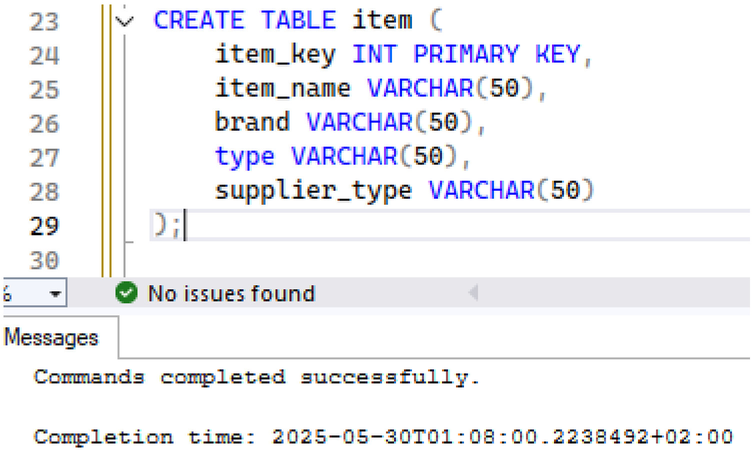
Branch dimension



```
CREATE TABLE branch (
    branch_key INT PRIMARY KEY,
    branch_name VARCHAR(50),
    branch_type VARCHAR(50)
);
```

```
Commands completed successfully.

Completion time: 2025-05-30T01:06:24.6596790+02:00
```

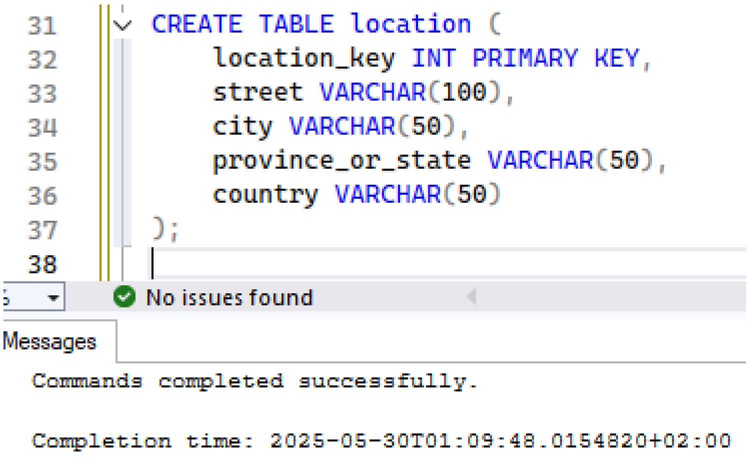
Item dimension



```
CREATE TABLE item (
    item_key INT PRIMARY KEY,
    item_name VARCHAR(50),
    brand VARCHAR(50),
    type VARCHAR(50),
    supplier_type VARCHAR(50)
);
```

```
Commands completed successfully.

Completion time: 2025-05-30T01:08:00.2238492+02:00
```

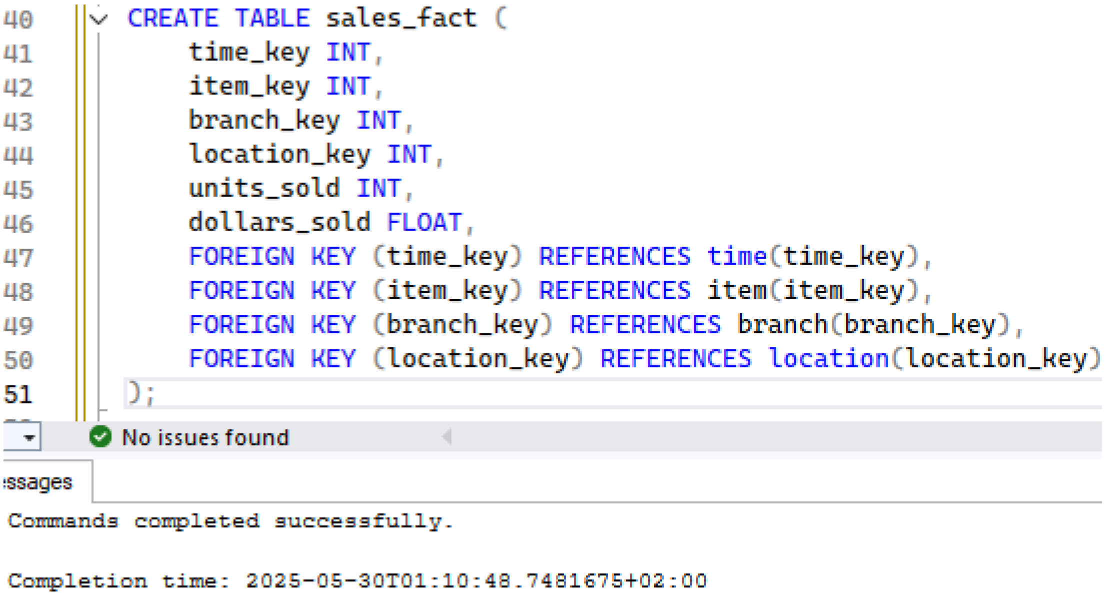
Location dimension



```sql
CREATE TABLE location (
    location_key INT PRIMARY KEY,
    street VARCHAR(100),
    city VARCHAR(50),
    province_or_state VARCHAR(50),
    country VARCHAR(50)
);
```

```
Commands completed successfully.

Completion time: 2025-05-30T01:09:48.0154820+02:00
```

Sales Fact table

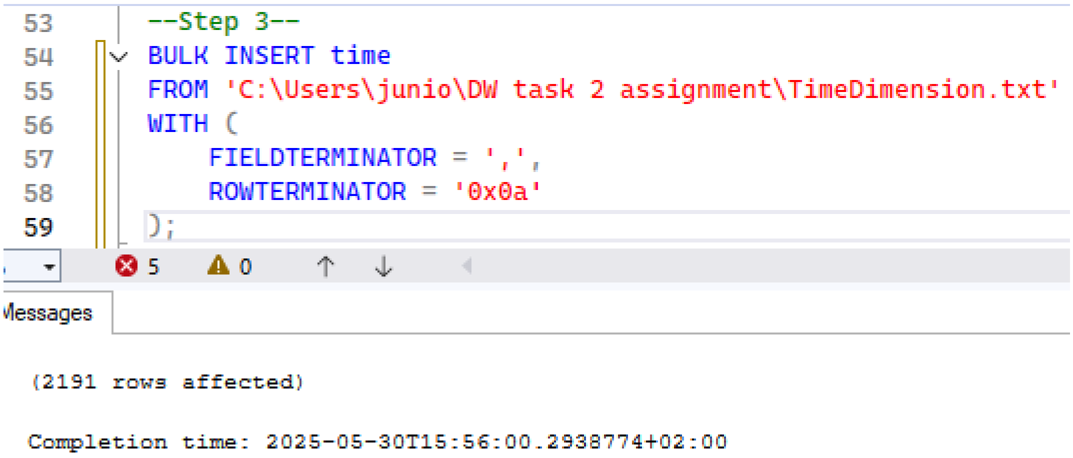


```sql
CREATE TABLE sales_fact (
    time_key INT,
    item_key INT,
    branch_key INT,
    location_key INT,
    units_sold INT,
    dollars_sold FLOAT,
    FOREIGN KEY (time_key) REFERENCES time(time_key),
    FOREIGN KEY (item_key) REFERENCES item(item_key),
    FOREIGN KEY (branch_key) REFERENCES branch(branch_key),
    FOREIGN KEY (location_key) REFERENCES location(location_key)
);
```

```
Commands completed successfully.

Completion time: 2025-05-30T01:10:48.7481675+02:00
```

STEP 3 APPROPRIATE DATA

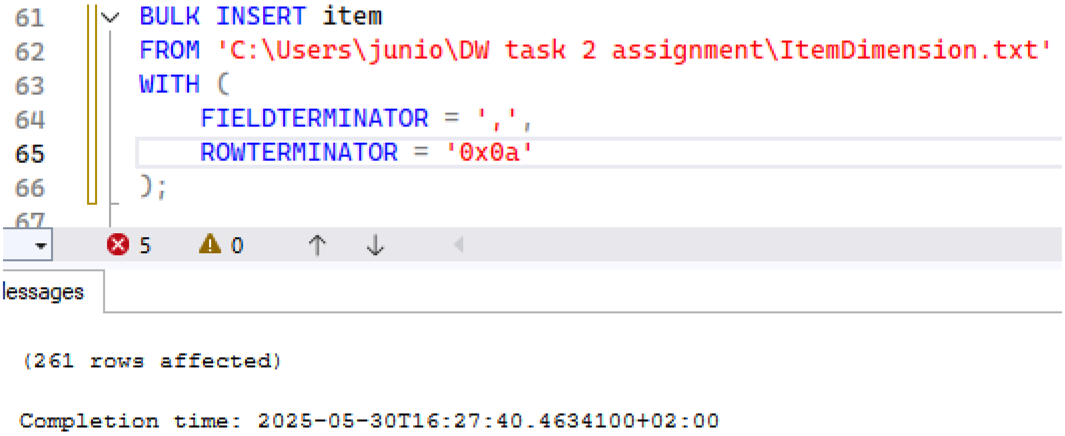
Time data



```sql
--Step 3--
BULK INSERT time
FROM 'C:\Users\junio\DW task 2 assignment\TimeDimension.txt'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a'
);
```

```
(2191 rows affected)

Completion time: 2025-05-30T15:56:00.2938774+02:00
```

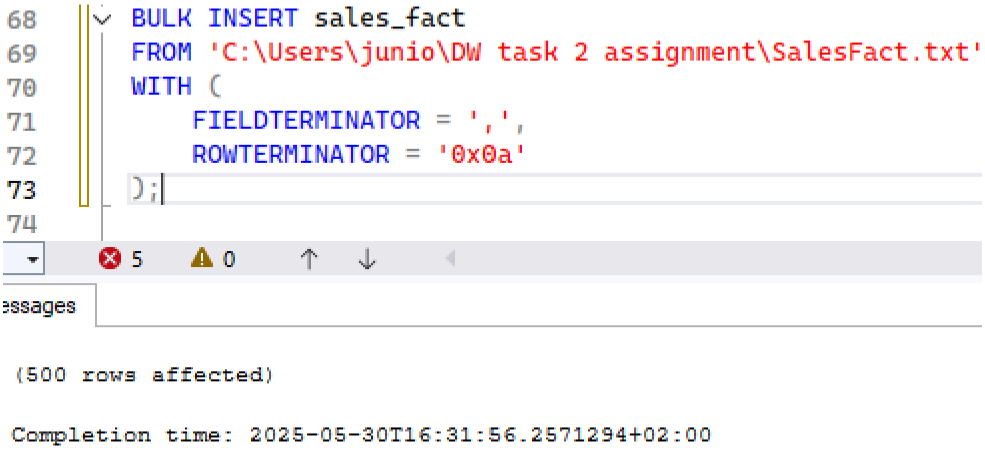
Item data



```
BULK INSERT item
FROM 'C:\Users\junio\DW task 2 assignment\ItemDimension.txt'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a'
);
```

```
(261 rows affected)

Completion time: 2025-05-30T16:27:40.4634100+02:00
```

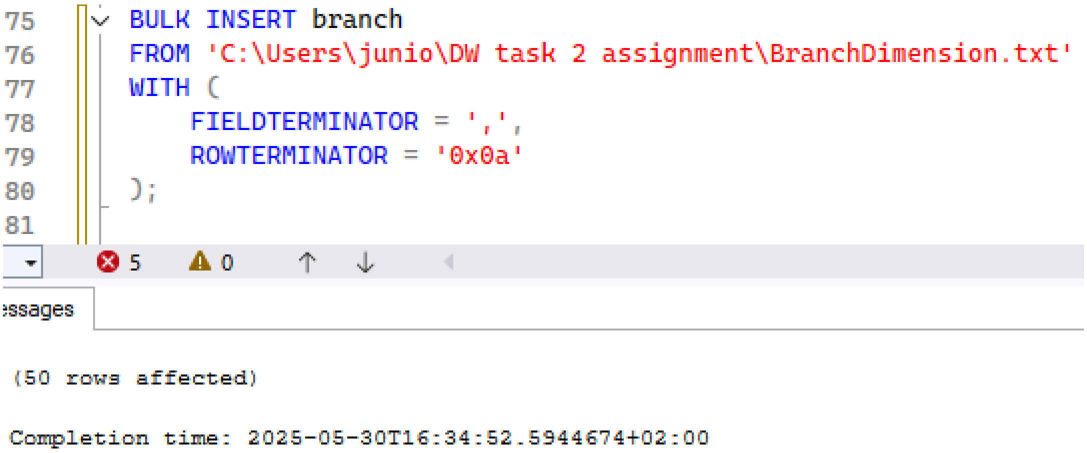
Sales_fact data



```
BULK INSERT sales_fact
FROM 'C:\Users\junio\DW task 2 assignment\SalesFact.txt'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a'
);
```

```
(500 rows affected)

Completion time: 2025-05-30T16:31:56.2571294+02:00
```

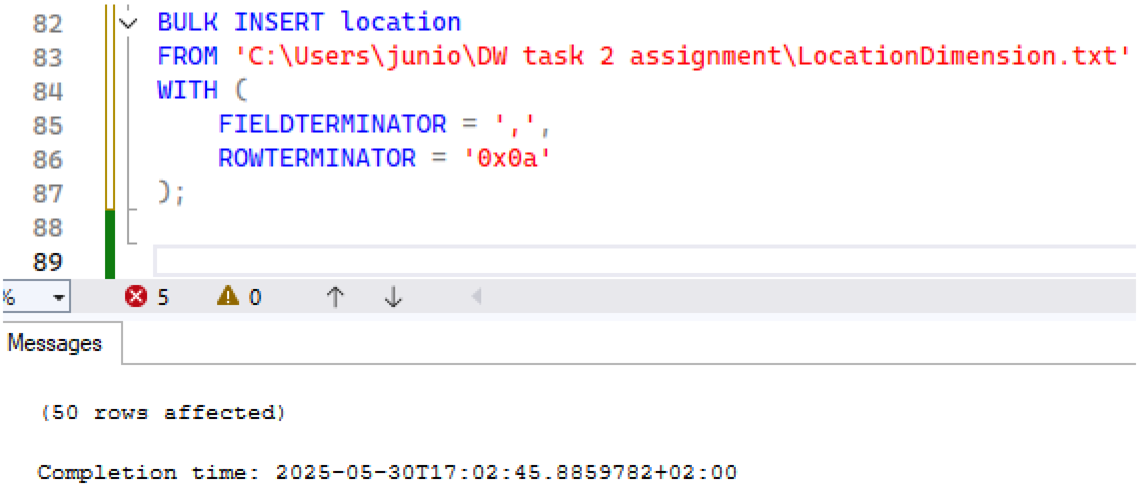
Branch data



```
BULK INSERT branch
FROM 'C:\Users\junio\DW task 2 assignment\BranchDimension.txt'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a'
);
```

```
(50 rows affected)

Completion time: 2025-05-30T16:34:52.5944674+02:00
```

Location data

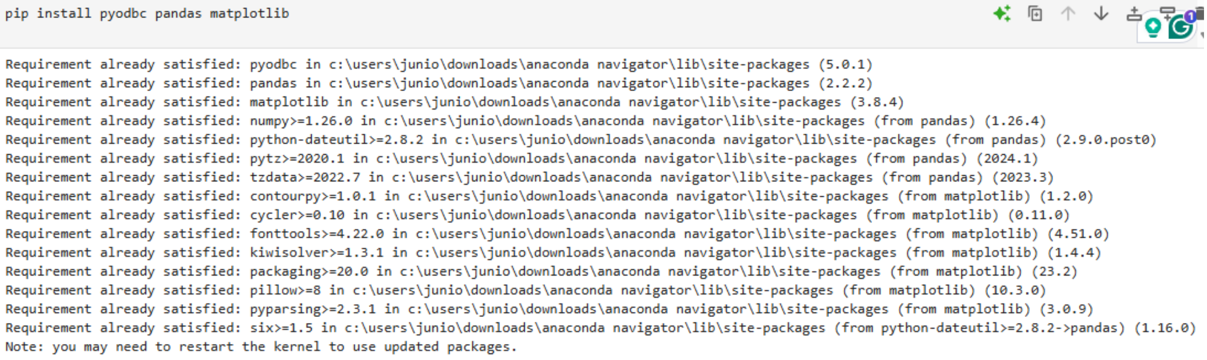


```sql
BULK INSERT location
FROM 'C:\Users\junio\DW task 2 assignment\LocationDimension.txt'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a'
);
```

```
(50 rows affected)

Completion time: 2025-05-30T17:02:45.8859782+02:00
```

STEP 4: CONNECTION OF SQL SERVER MANAGEMENT STUDIO TO PYTHON

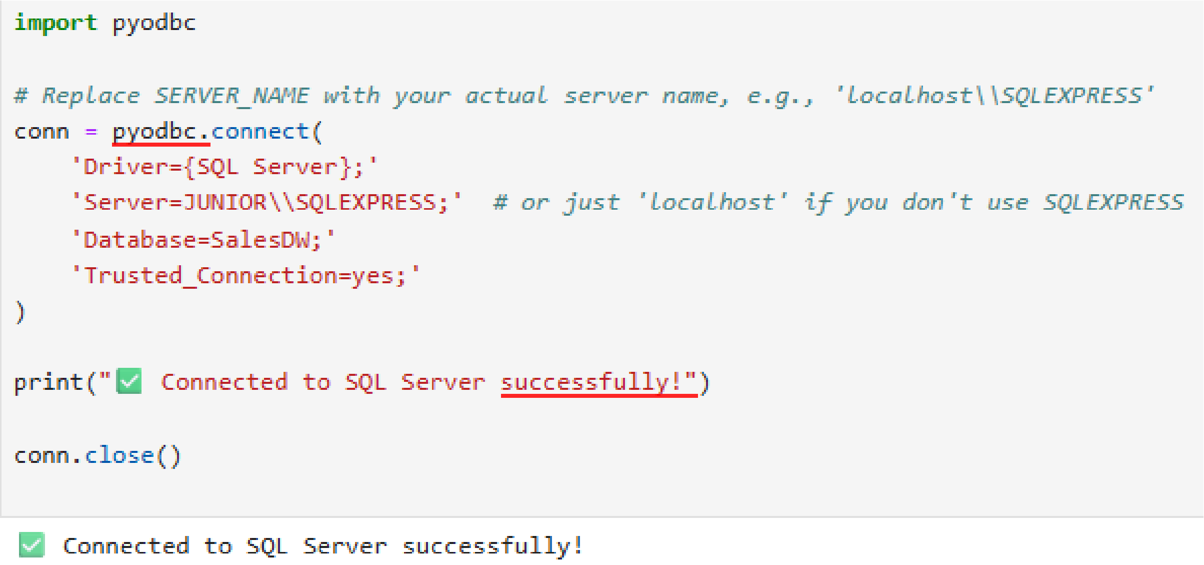
Installing pyodbc



```
pip install pyodbc pandas matplotlib
```

```
Requirement already satisfied: pyodbc in c:\users\junio\downloads\anaconda navigator\lib\site-packages (5.0.1)
Requirement already satisfied: pandas in c:\users\junio\downloads\anaconda navigator\lib\site-packages (2.2.2)
Requirement already satisfied: matplotlib in c:\users\junio\downloads\anaconda navigator\lib\site-packages (3.8.4)
Requirement already satisfied: numpy>=1.26.0 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\users\junio\downloads\anaconda navigator\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Making SQL connection to python

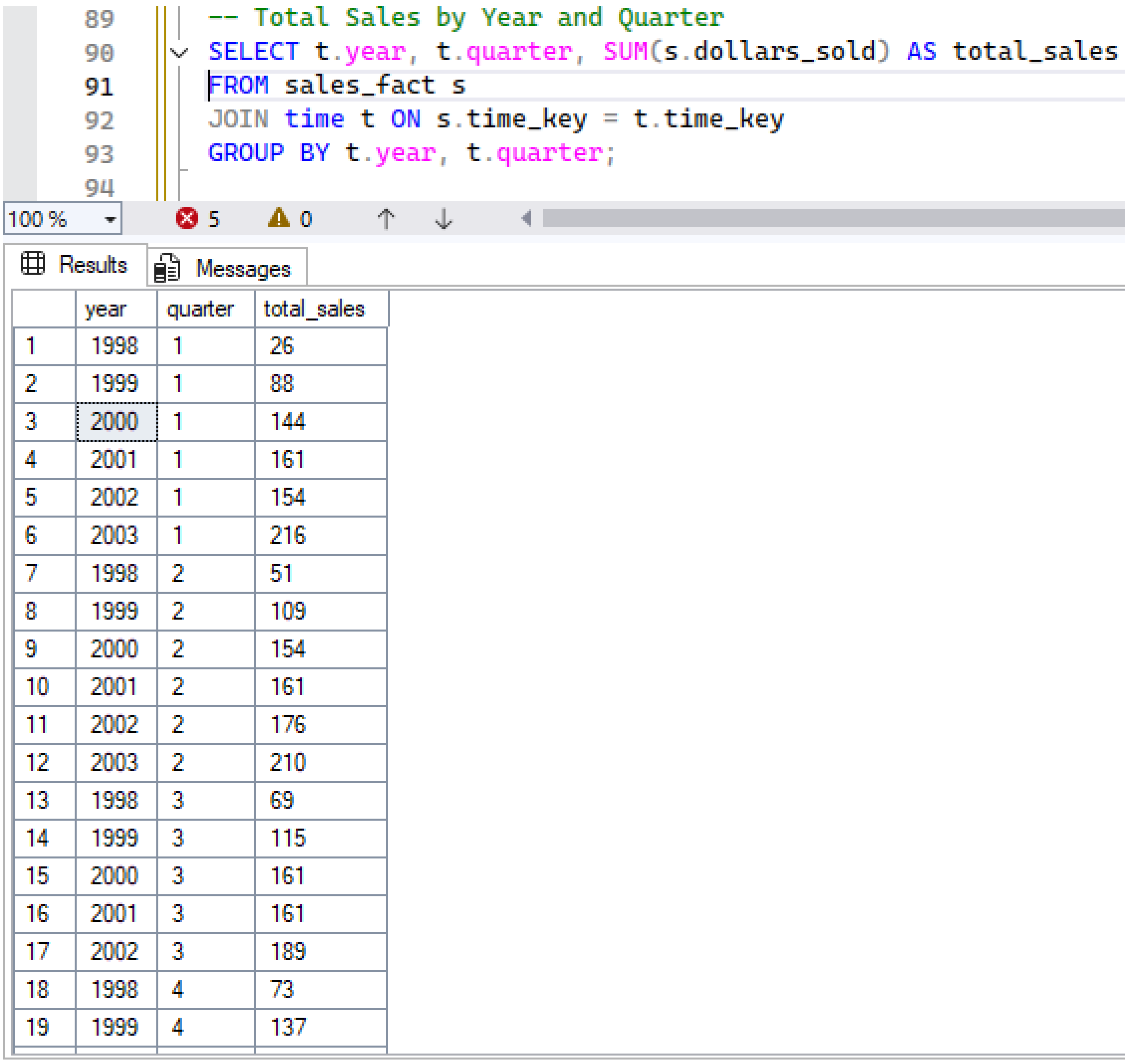


```
import pyodbc

# Replace SERVER_NAME with your actual server name, e.g., 'localhost\\SQLEXPRESS'
conn = pyodbc.connect(
    'Driver={SQL Server};'
    'Server=JUNIOR\\SQLEXPRESS;'  # or just 'localhost' if you don't use SQLEXPRESS
    'Database=SalesDW;'
    'Trusted_Connection=yes;'
)

print("✅ Connected to SQL Server successfully!")

conn.close()
```

```
✅ Connected to SQL Server successfully!
```

STEP 5-6: IMPLEMENT ROLL-UP AND DRILL-DOWN QUERIES

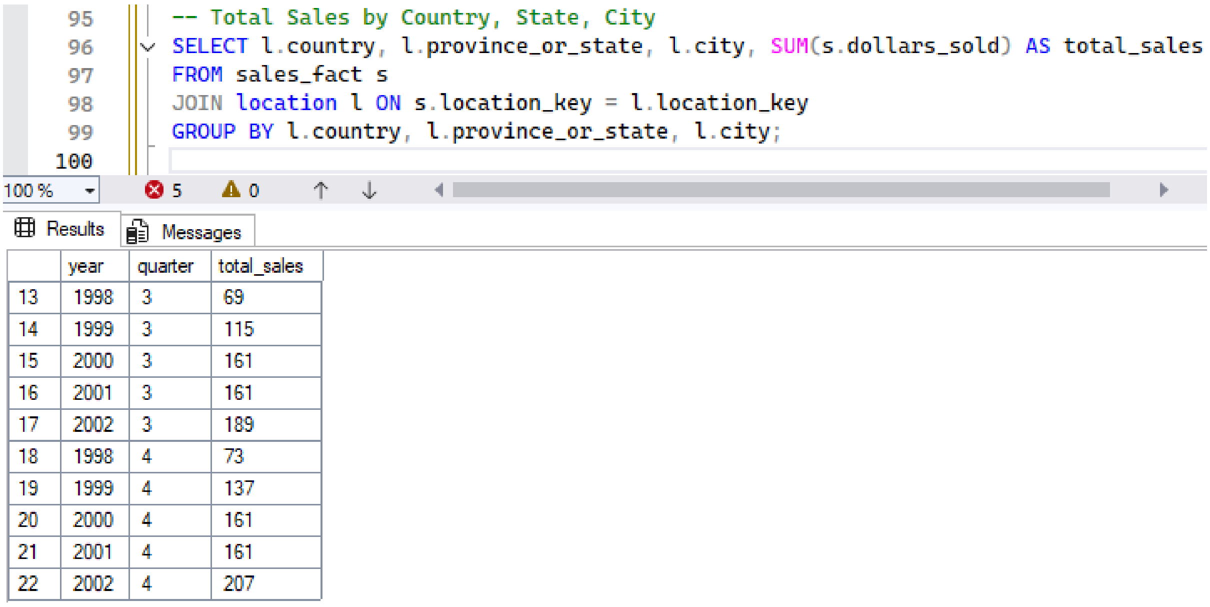
Rolling up total sales by Year and Quarter



```
-- Total Sales by Year and Quarter
SELECT t.year, t.quarter, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN time t ON s.time_key = t.time_key
GROUP BY t.year, t.quarter;
```

	year	quarter	total_sales
1	1998	1	26
2	1999	1	88
3	2000	1	144
4	2001	1	161
5	2002	1	154
6	2003	1	216
7	1998	2	51
8	1999	2	109
9	2000	2	154
10	2001	2	161
11	2002	2	176
12	2003	2	210
13	1998	3	69
14	1999	3	115
15	2000	3	161
16	2001	3	161
17	2002	3	189
18	1998	4	73
19	1999	4	137

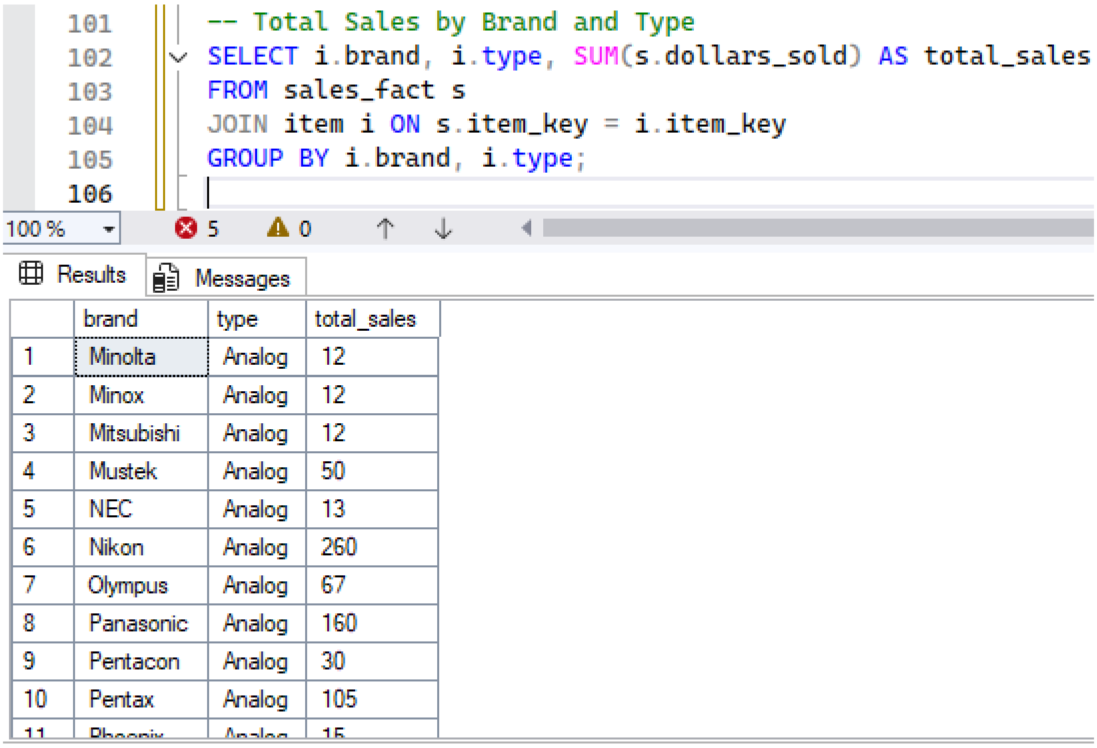
Rolling up total sales by Country, State and City



```
-- Total Sales by Country, State, City
SELECT l.country, l.province_or_state, l.city, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN location l ON s.location_key = l.location_key
GROUP BY l.country, l.province_or_state, l.city;
```

	year	quarter	total_sales
13	1998	3	69
14	1999	3	115
15	2000	3	161
16	2001	3	161
17	2002	3	189
18	1998	4	73
19	1999	4	137
20	2000	4	161
21	2001	4	161
22	2002	4	207

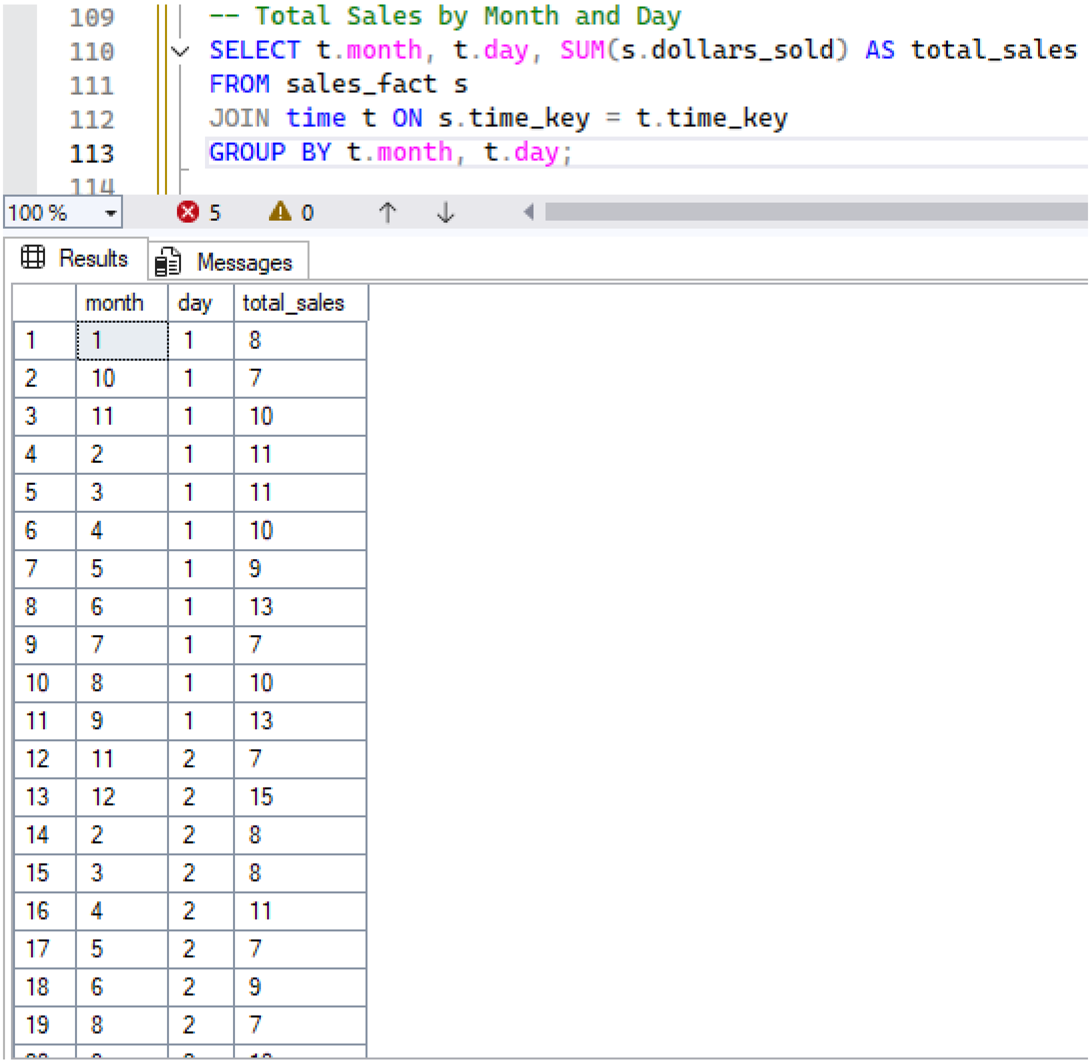
Rolling up total sales by Brand and Type



```sql
-- Total Sales by Brand and Type
SELECT i.brand, i.type, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN item i ON s.item_key = i.item_key
GROUP BY i.brand, i.type;
```

	brand	type	total_sales
1	Minolta	Analog	12
2	Minox	Analog	12
3	Mitsubishi	Analog	12
4	Mustek	Analog	50
5	NEC	Analog	13
6	Nikon	Analog	260
7	Olympus	Analog	67
8	Panasonic	Analog	160
9	Pentacon	Analog	30
10	Pentax	Analog	105
11	Phoenix	Analog	15

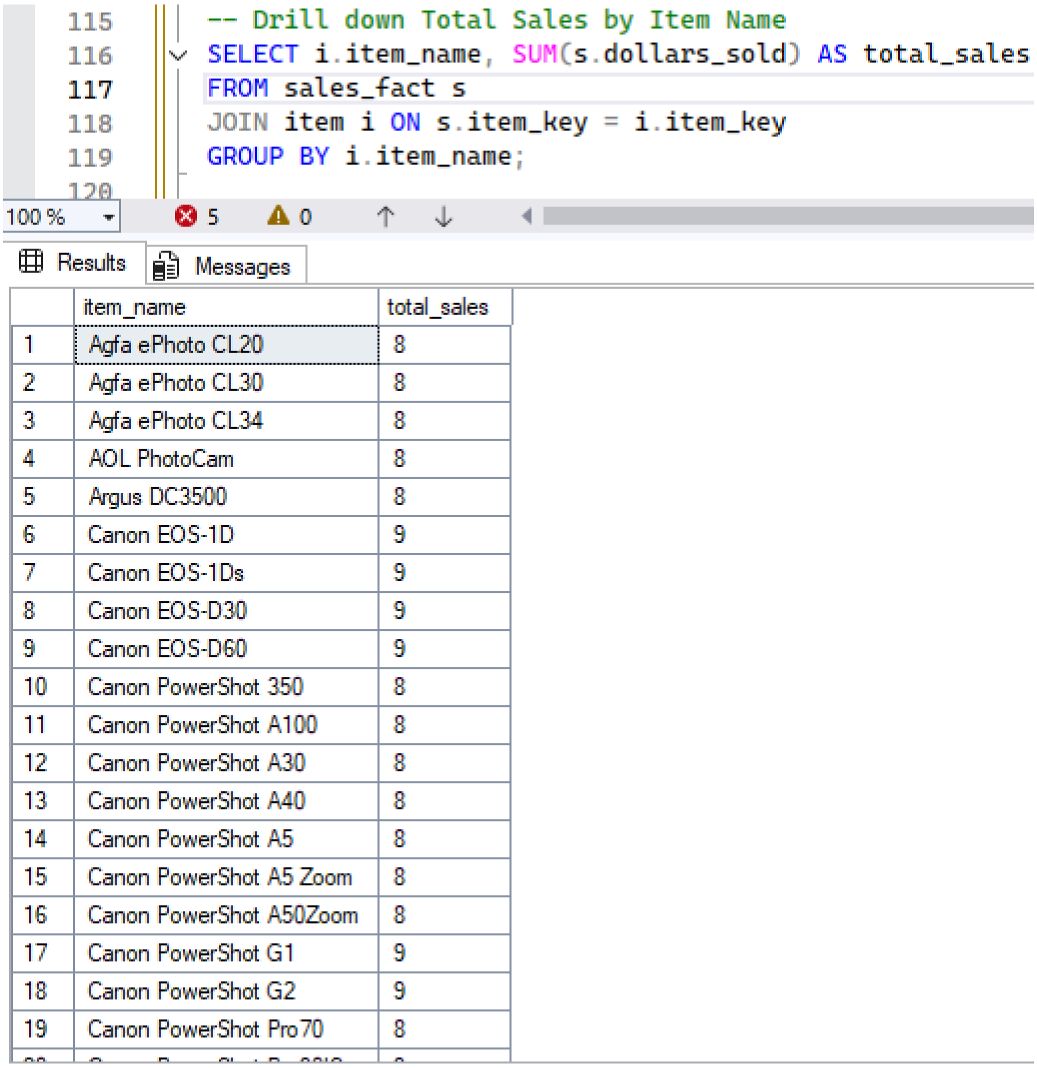
Drill down total sales by Month and Day



```sql
-- Total Sales by Month and Day
SELECT t.month, t.day, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN time t ON s.time_key = t.time_key
GROUP BY t.month, t.day;
```

	month	day	total_sales
1	1	1	8
2	10	1	7
3	11	1	10
4	2	1	11
5	3	1	11
6	4	1	10
7	5	1	9
8	6	1	13
9	7	1	7
10	8	1	10
11	9	1	13
12	11	2	7
13	12	2	15
14	2	2	8
15	3	2	8
16	4	2	11
17	5	2	7
18	6	2	9
19	8	2	7

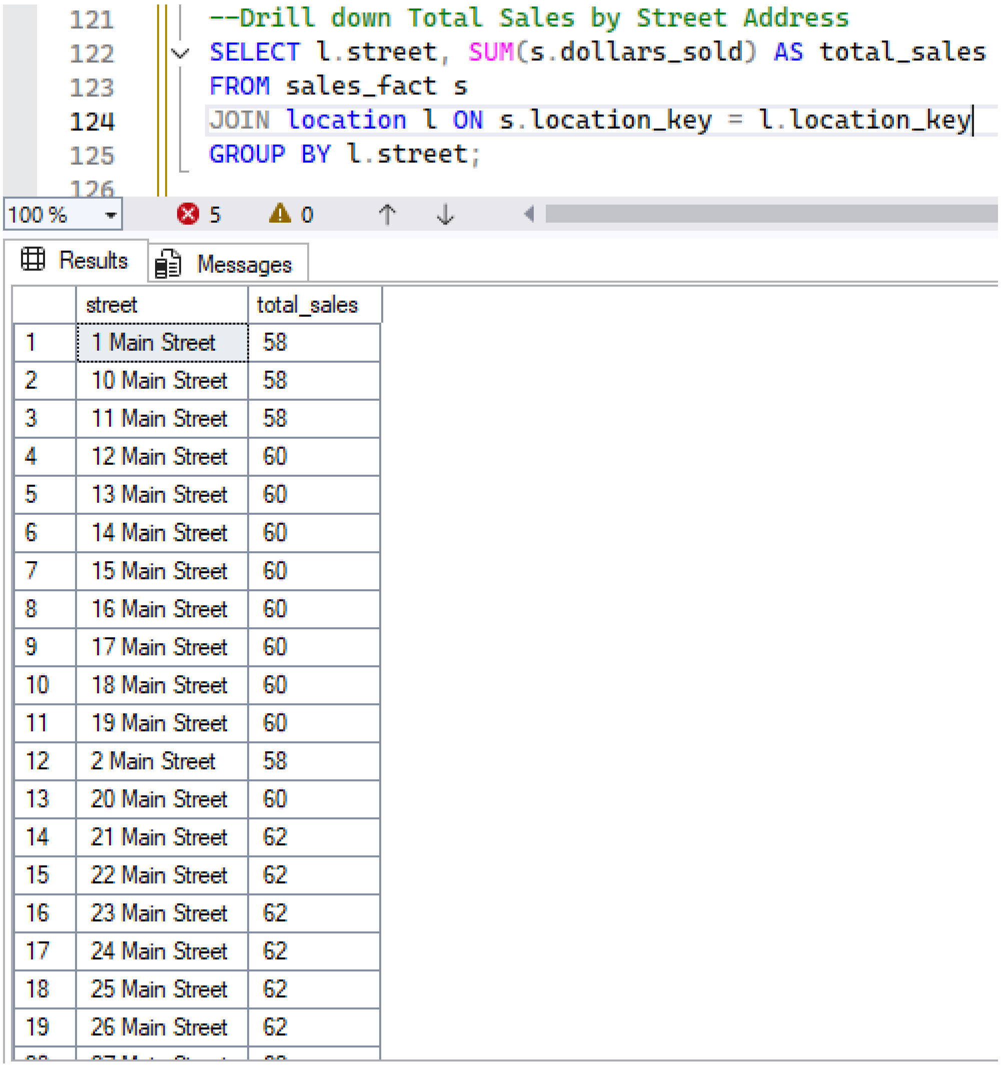
Drill down total sales by item name



```
-- Drill down Total Sales by Item Name
SELECT i.item_name, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN item i ON s.item_key = i.item_key
GROUP BY i.item_name;
```

	item_name	total_sales
1	Agfa ePhoto CL20	8
2	Agfa ePhoto CL30	8
3	Agfa ePhoto CL34	8
4	AOL PhotoCam	8
5	Argus DC3500	8
6	Canon EOS-1D	9
7	Canon EOS-1Ds	9
8	Canon EOS-D30	9
9	Canon EOS-D60	9
10	Canon PowerShot 350	8
11	Canon PowerShot A100	8
12	Canon PowerShot A30	8
13	Canon PowerShot A40	8
14	Canon PowerShot A5	8
15	Canon PowerShot A5 Zoom	8
16	Canon PowerShot A50Zoom	8
17	Canon PowerShot G1	9
18	Canon PowerShot G2	9
19	Canon PowerShot Pro70	8

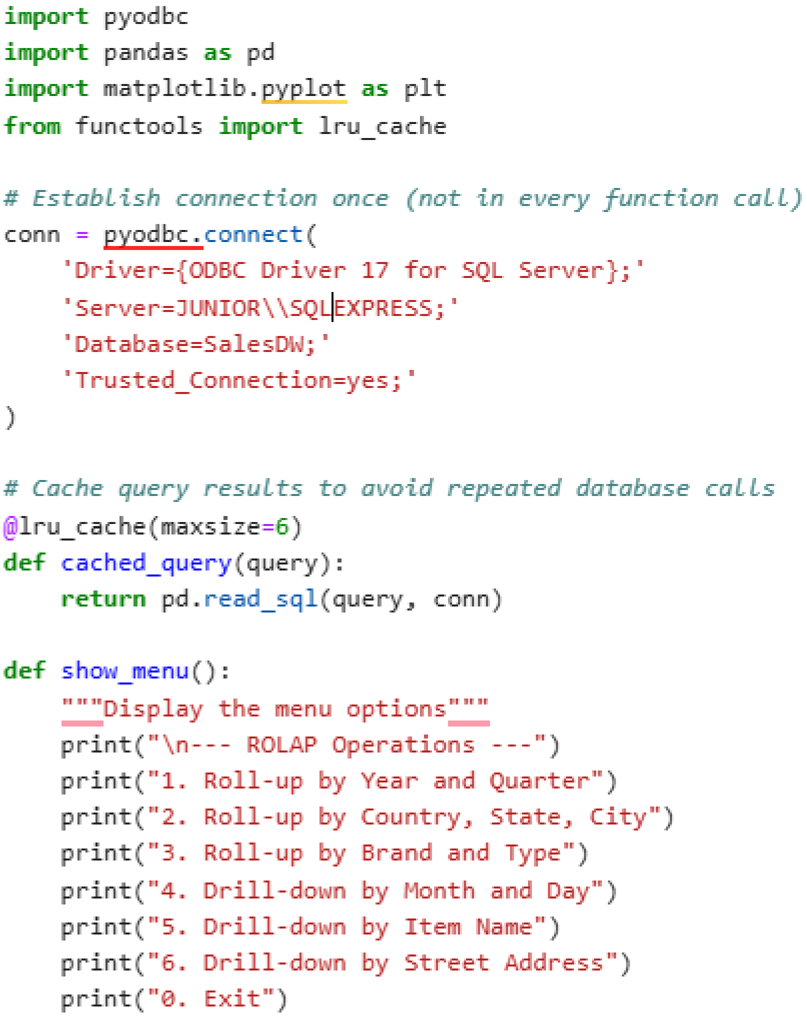
Drill down total sales by Street Address



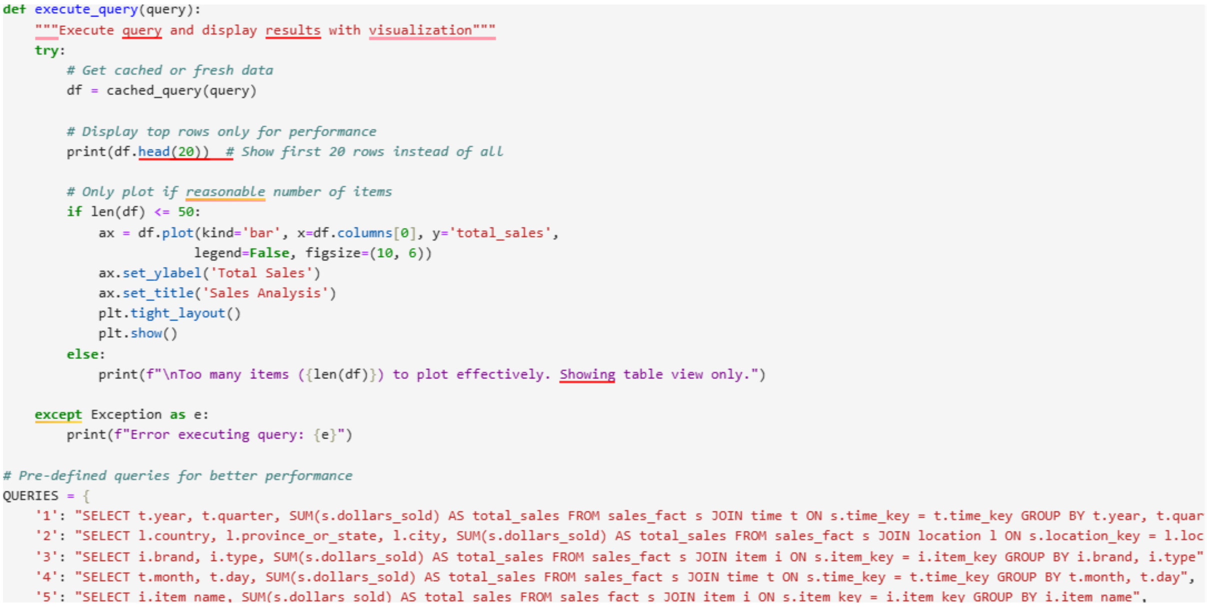
```sql
--Drill down Total Sales by Street Address
SELECT l.street, SUM(s.dollars_sold) AS total_sales
FROM sales_fact s
JOIN location l ON s.location_key = l.location_key
GROUP BY l.street;
```

	street	total_sales
1	1 Main Street	58
2	10 Main Street	58
3	11 Main Street	58
4	12 Main Street	60
5	13 Main Street	60
6	14 Main Street	60
7	15 Main Street	60
8	16 Main Street	60
9	17 Main Street	60
10	18 Main Street	60
11	19 Main Street	60
12	2 Main Street	58
13	20 Main Street	60
14	21 Main Street	62
15	22 Main Street	62
16	23 Main Street	62
17	24 Main Street	62
18	25 Main Street	62
19	26 Main Street	62

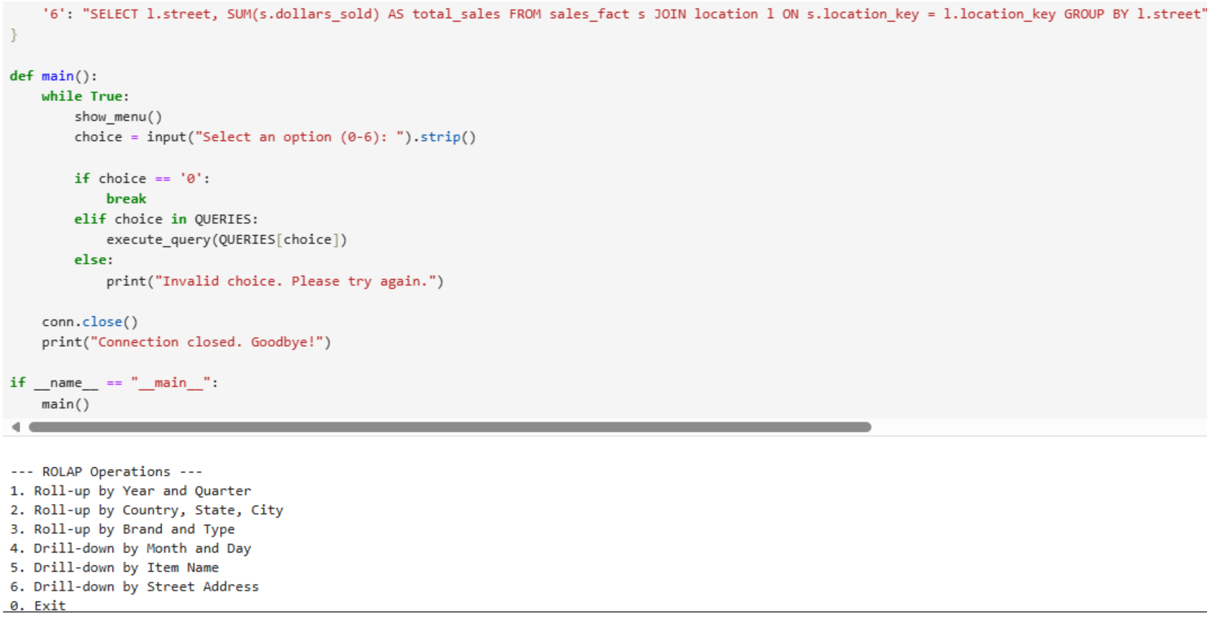
STEP 7: SIMPLE USER INTERFACE



```python
import pyodbc
import pandas as pd
import matplotlib.pyplot as plt
from functools import lru_cache

# Establish connection once (not in every function call)
conn = pyodbc.connect(
    'Driver={ODBC Driver 17 for SQL Server};'
    'Server=JUNIOR\\SQLEXPRESS;'
    'Database=SalesDW;'
    'Trusted_Connection=yes;'
)

# Cache query results to avoid repeated database calls
@lru_cache(maxsize=6)
def cached_query(query):
    return pd.read_sql(query, conn)

def show_menu():
    """Display the menu options"""
    print("\n--- ROLAP Operations ---")
    print("1. Roll-up by Year and Quarter")
    print("2. Roll-up by Country, State, City")
    print("3. Roll-up by Brand and Type")
    print("4. Drill-down by Month and Day")
    print("5. Drill-down by Item Name")
    print("6. Drill-down by Street Address")
    print("0. Exit")
```



```python
def execute_query(query):
    """Execute query and display results with visualization"""
    try:
        # Get cached or fresh data
        df = cached_query(query)

        # Display top rows only for performance
        print(df.head(20))  # Show first 20 rows instead of all

        # Only plot if reasonable number of items
        if len(df) <= 50:
            ax = df.plot(kind='bar', x=df.columns[0], y='total_sales',
                         legend=False, figsize=(10, 6))
            ax.set_ylabel('Total Sales')
            ax.set_title('Sales Analysis')
            plt.tight_layout()
            plt.show()
        else:
            print(f"\nToo many items ({len(df)}) to plot effectively. Showing table view only.")

    except Exception as e:
        print(f"Error executing query: {e}")

# Pre-defined queries for better performance
QUERIES = {
    '1': "SELECT t.year, t.quarter, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN time t ON s.time_key = t.time_key GROUP BY t.year, t.quar
    '2': "SELECT l.country, l.province_or_state, l.city, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN location l ON s.location_key = l.loc
    '3': "SELECT i.brand, i.type, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN item i ON s.item_key = i.item_key GROUP BY i.brand, i.type"
    '4': "SELECT t.month, t.day, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN time t ON s.time_key = t.time_key GROUP BY t.month, t.day",
    '5': "SELECT i.item_name, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN item i ON s.item_key = i.item_key GROUP BY i.item_name",
```



```
    '6': "SELECT l.street, SUM(s.dollars_sold) AS total_sales FROM sales_fact s JOIN location l ON s.location_key = l.location_key GROUP BY l.street"
}

def main():
    while True:
        show_menu()
        choice = input("Select an option (0-6): ").strip()

        if choice == '0':
            break
        elif choice in QUERIES:
            execute_query(QUERIES[choice])
        else:
            print("Invalid choice. Please try again.")

    conn.close()
    print("Connection closed. Goodbye!")

if __name__ == "__main__":
    main()
```

```
--- ROLAP Operations ---
1. Roll-up by Year and Quarter
2. Roll-up by Country, State, City
3. Roll-up by Brand and Type
4. Drill-down by Month and Day
5. Drill-down by Item Name
6. Drill-down by Street Address
0. Exit
```

STEP 8: VISUALIZATION OF RESULTS

Diagram for Roll-up by Year and Quarter

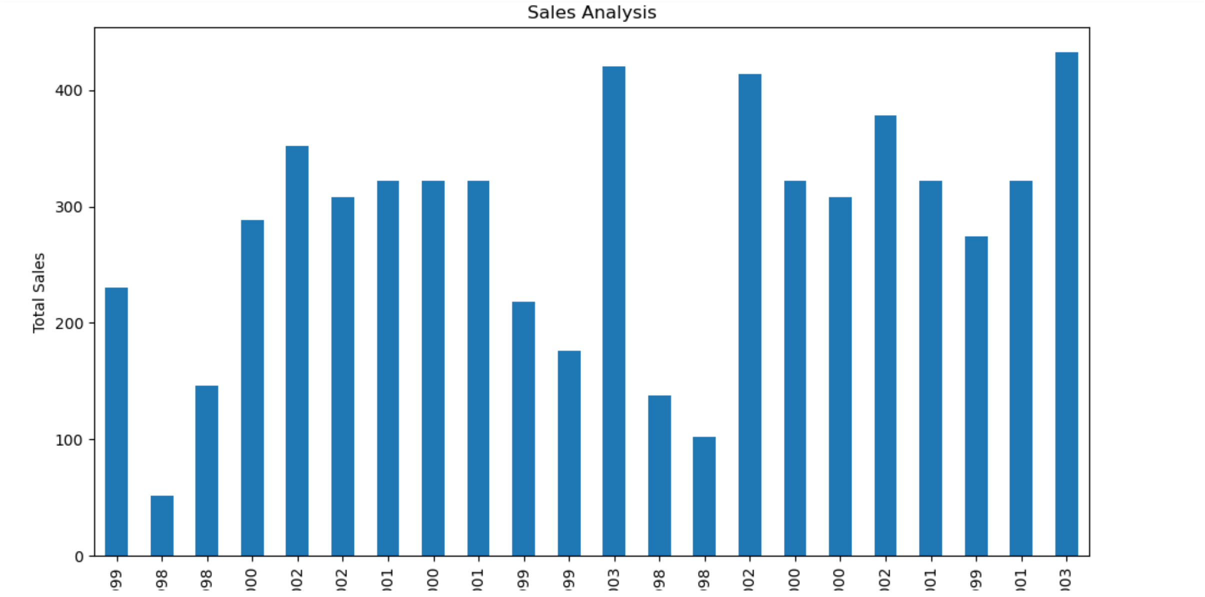
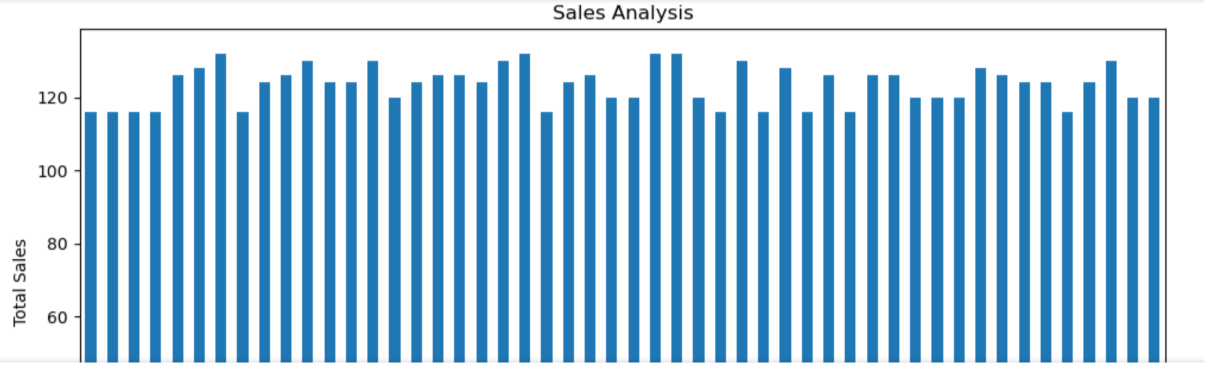


Diagram for Roll-up by Country, State, City



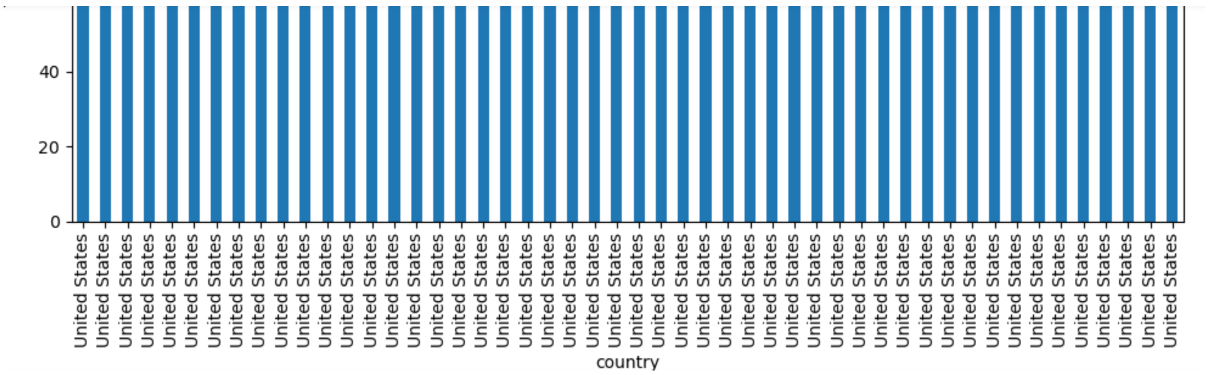


Diagram for Roll-up by Brand and Type

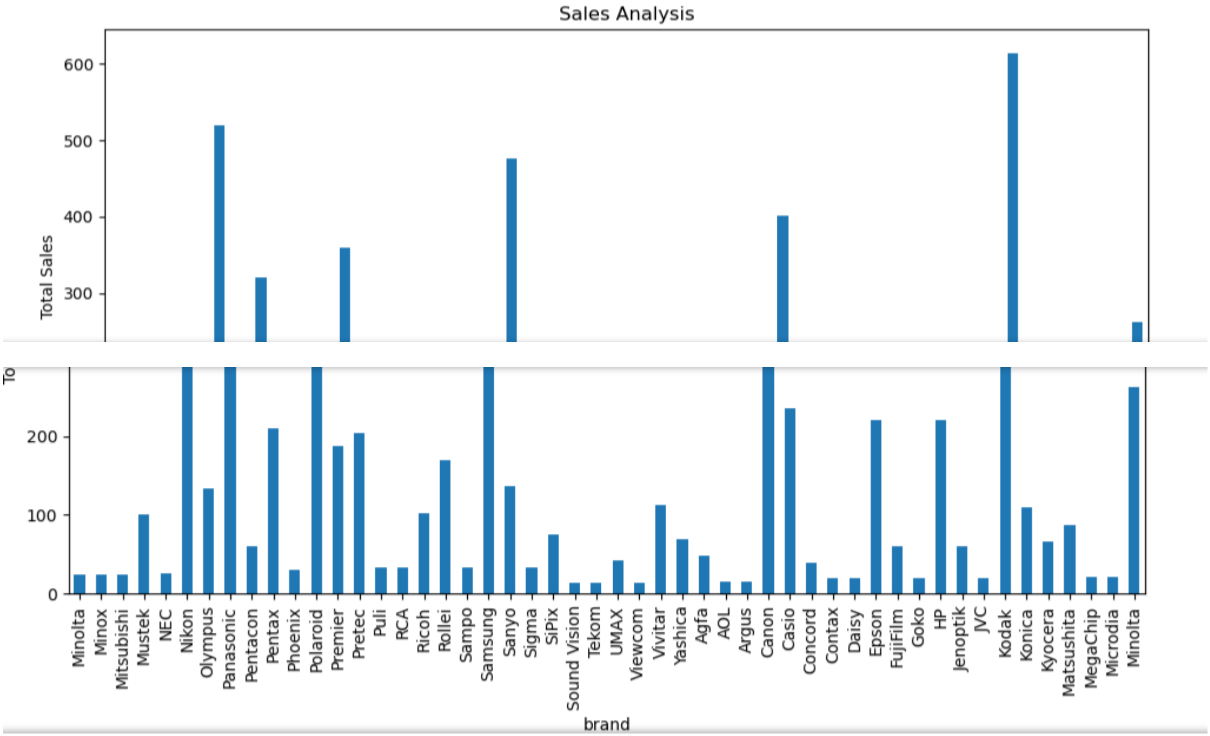
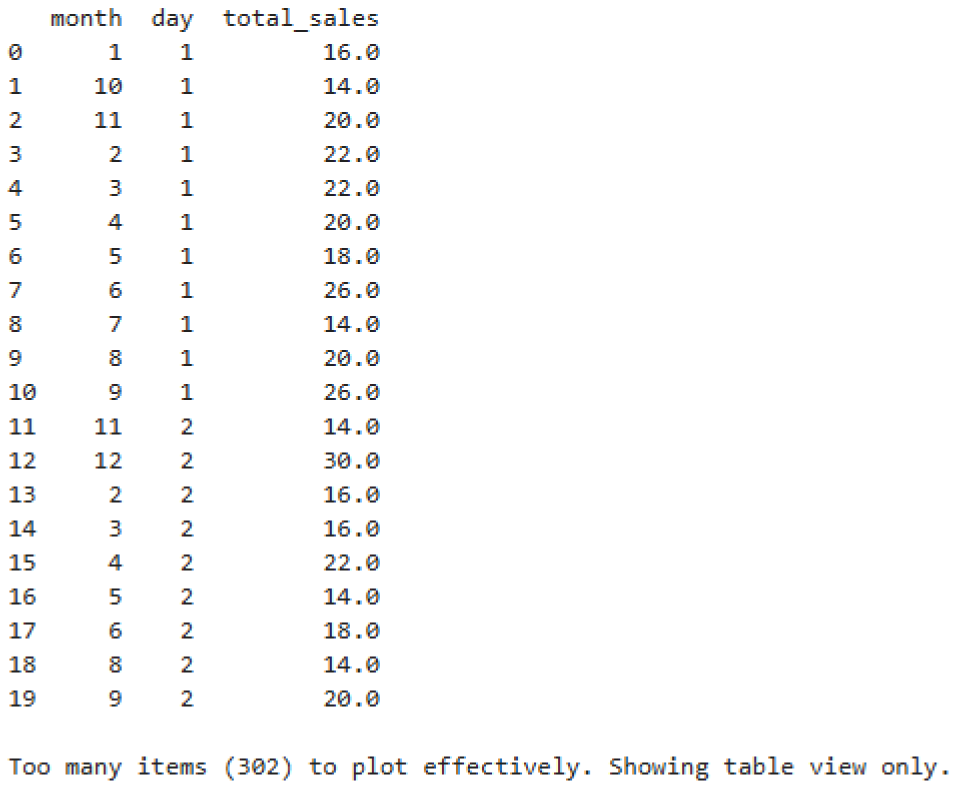
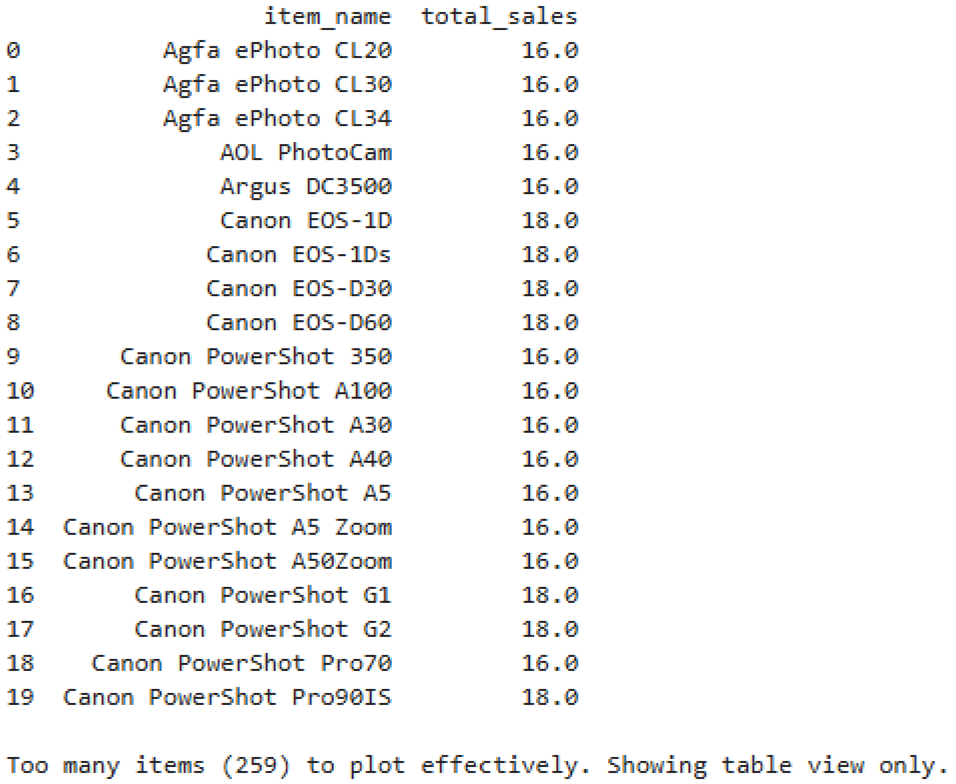


Diagram for Drill-down by Month and Day



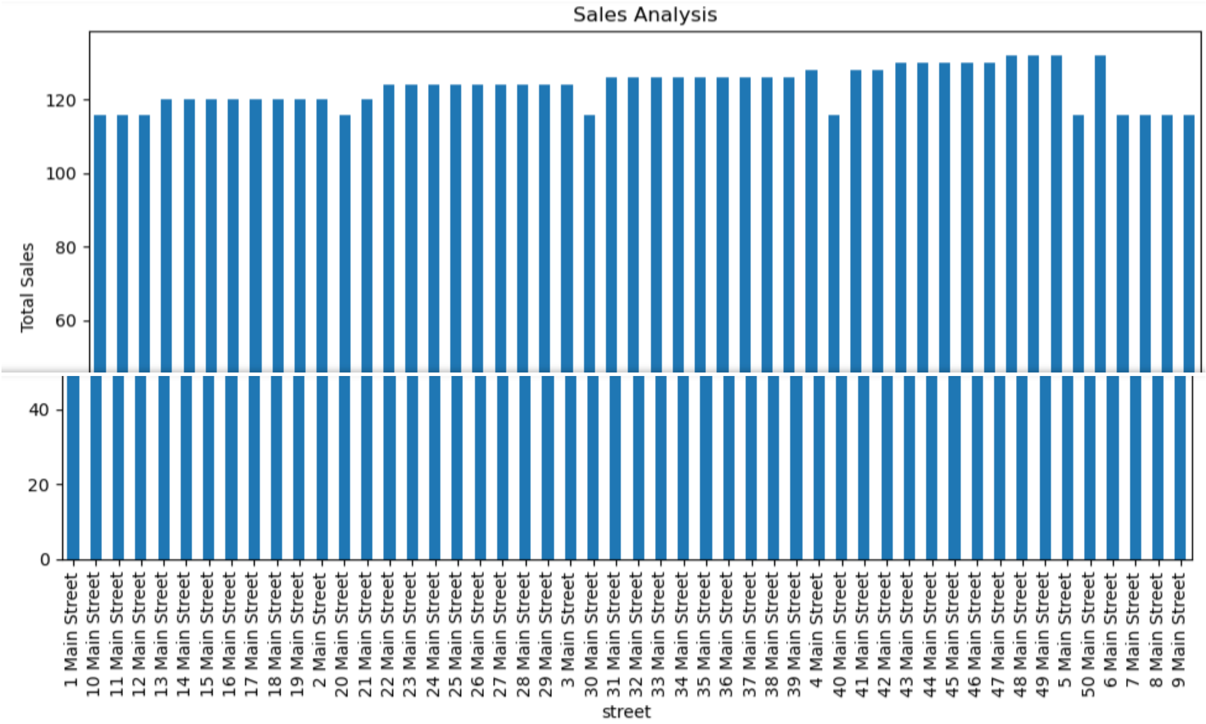
```
   month  day  total_sales
0      1    1         16.0
1     10    1         14.0
2     11    1         20.0
3      2    1         22.0
4      3    1         22.0
5      4    1         20.0
6      5    1         18.0
7      6    1         26.0
8      7    1         14.0
9      8    1         20.0
10     9    1         26.0
11    11    2         14.0
12    12    2         30.0
13     2    2         16.0
14     3    2         16.0
15     4    2         22.0
16     5    2         14.0
17     6    2         18.0
18     8    2         14.0
19     9    2         20.0

Too many items (302) to plot effectively. Showing table view only.
```

Diagram for Drill-down by Item Name



```
                 item_name  total_sales
0          Agfa ePhoto CL20         16.0
1          Agfa ePhoto CL30         16.0
2          Agfa ePhoto CL34         16.0
3              AOL PhotoCam         16.0
4              Argus DC3500         16.0
5              Canon EOS-1D         18.0
6             Canon EOS-1Ds         18.0
7             Canon EOS-D30         18.0
8             Canon EOS-D60         18.0
9       Canon PowerShot 350         16.0
10     Canon PowerShot A100         16.0
11      Canon PowerShot A30         16.0
12      Canon PowerShot A40         16.0
13       Canon PowerShot A5         16.0
14  Canon PowerShot A5 Zoom         16.0
15  Canon PowerShot A50Zoom         16.0
16       Canon PowerShot G1         18.0
17       Canon PowerShot G2         18.0
18    Canon PowerShot Pro70         16.0
19  Canon PowerShot Pro90IS         18.0

Too many items (259) to plot effectively. Showing table view only.
```

Diagram for Drill-down by Street Address


Sales Analysis
Total Sales
120
100
80
60
40
20
0
1 Main Street
10 Main Street
11 Main Street
12 Main Street
13 Main Street
14 Main Street
15 Main Street
16 Main Street
17 Main Street
18 Main Street
19 Main Street
2 Main Street
20 Main Street
21 Main Street
22 Main Street
23 Main Street
24 Main Street
25 Main Street
26 Main Street
27 Main Street
28 Main Street
29 Main Street
3 Main Street
30 Main Street
31 Main Street
32 Main Street
33 Main Street
34 Main Street
35 Main Street
36 Main Street
37 Main Street
38 Main Street
39 Main Street
4 Main Street
40 Main Street
41 Main Street
42 Main Street
43 Main Street
44 Main Street
45 Main Street
46 Main Street
47 Main Street
48 Main Street
49 Main Street
5 Main Street
50 Main Street
6 Main Street
7 Main Street
8 Main Street
9 Main Street
street