Practical Workbook
SE-407
Data Warehouse & Mining

Name : ____________________________

Year : ____________________________

Batch : ____________________________

Roll No :

____

Department:_________________________

___

Department of Software Engineering NED University of Engineering & Technology

TABLE OF CONTENTS

Lab Session 01

Introduction to Data Warehouse, Data Mining, and tools Installation

Background

A Data Warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process. Unlike Online Transaction Processing (OLTP) systems optimized for transactions, Online Analytical Processing (OLAP) systems (like data warehouses) are optimized for complex queries and analysis. SQL (Structured Query Language) is the standard language for interacting with relational databases, which often form the foundation of data warehouses.

What is a Data Warehouse?

A **data warehouse** is a centralized repository that integrates data from multiple sources (e.g., databases, files) to support analytical reporting, structured queries, and decision-making.
Key Characteristics:

- **Subject-Oriented**: Organized around business subjects (e.g., sales, customers). • **Integrated**: Data is cleaned, transformed, and standardized.
- **Time-Variant**: Historical data stored for trend analysis.
- **Non-Volatile**: Data is read-only and not frequently updated.

Example:
A retail company uses a data warehouse to analyze sales trends across regions over the

past 5 years. **What is Data Mining?**

Data mining is the process of discovering hidden patterns, correlations, and insights from large datasets using techniques like clustering, classification, and association rule mining.
You can use command options to fine-tune the actions of a Linux command. Instead of making you learn a second command, Linux lets you modify the basic, or default, actions of the command by using options. Linux commands often use parameters that are not actual command options. These parameters, such as filenames or directories, are not preceded by a dash.
Common Techniques:

- **Classification**: Predicting categories (e.g., spam detection).
- **Clustering**: Grouping similar data points (e.g., customer segmentation).
- **Association Rule Mining**: Finding relationships (e.g., "customers who buy X also buy Y"). **Example**:

A bank uses data mining to detect fraudulent transactions based on

unusual spending patterns. **Required Software:**

- **RDBMS: PostgreSQL** (Latest stable version). Download from https://www.postgresql.org/download/.

- **SQL Client: pgAdmin** (usually bundled with PostgreSQL) or **DBeaver** (Community Edition - universal client). These provide a graphical interface to interact with PostgreSQL. You can also use the command-line tool psql.

- **Data Mining Environment: Python 3.x** via **Anaconda Distribution**. Anaconda simplifies package management and includes Python, Jupyter Notebook/Lab, and common data science libraries. Download from https://www.anaconda.com/products/distribution.
  - **Key Python Libraries:** Ensure you have installed (usually via conda install <library_name> or pip install <library_name> in your Anaconda environment):
    - pandas: For data manipulation and analysis.
    - numpy: For numerical operations.
    - scikit-learn: For machine learning algorithms (classification, clustering, preprocessing).
    - matplotlib: For basic plotting.
    - seaborn: For enhanced statistical plotting.
    - mlxtend: For association rule mining (Apriori).
  - psycopg2-binary: To connect Python to PostgreSQL (if needed for advanced ETL).

Installation Setup

PostgreSQL Installation (Windows Environment):

1. Download the latest PostgreSQL installer from the official website: https://www.postgresql.org/download/windows/
2. Run the installer and follow the installation wizard.
3. Set the password for the PostgreSQL administrator account (postgres).
4. Complete the installation and verify by opening "pgAdmin", a graphical tool for database administration. Create a specific database for this course work (e.g., dw_labs).

Python Installation Using Anaconda:

1. Download Anaconda from https://www.anaconda.com/products/individual
2. Run the Anaconda installer and follow the installation steps.
3. Choose to add Anaconda to your PATH during installation for ease of use. 4. Verify the installation by opening the Anaconda Navigator and launching Jupyter Notebook or Spyder IDE.
5. Launch Anaconda Navigator or use the Anaconda Prompt. Create a dedicated environment for this course (recommended) (e.g., conda create -n dwm_env python=3.9 pandas numpy scikit-learn matplotlib seaborn mlxtend psycopg2- binary).

Activate it (conda activate dwm_env).

Verify Installations

PostgreSQL Verification (using pgAdmin):

1. Launch pgAdmin.
2. Connect to your PostgreSQL server instance (you might need the password set during installation).
3. In the object browser (left pane), find your server, expand 'Databases'. Right-click 'Databases' -> Create -> Database. Name it dw_labs.
4. Select the dw_labs database. Click the 'Tools' menu -> 'Query Tool'.
5. In the Query Editor window, type SELECT version(); and execute (click the 'Play' button or press F5). You should see the PostgreSQL version details in the output pane.

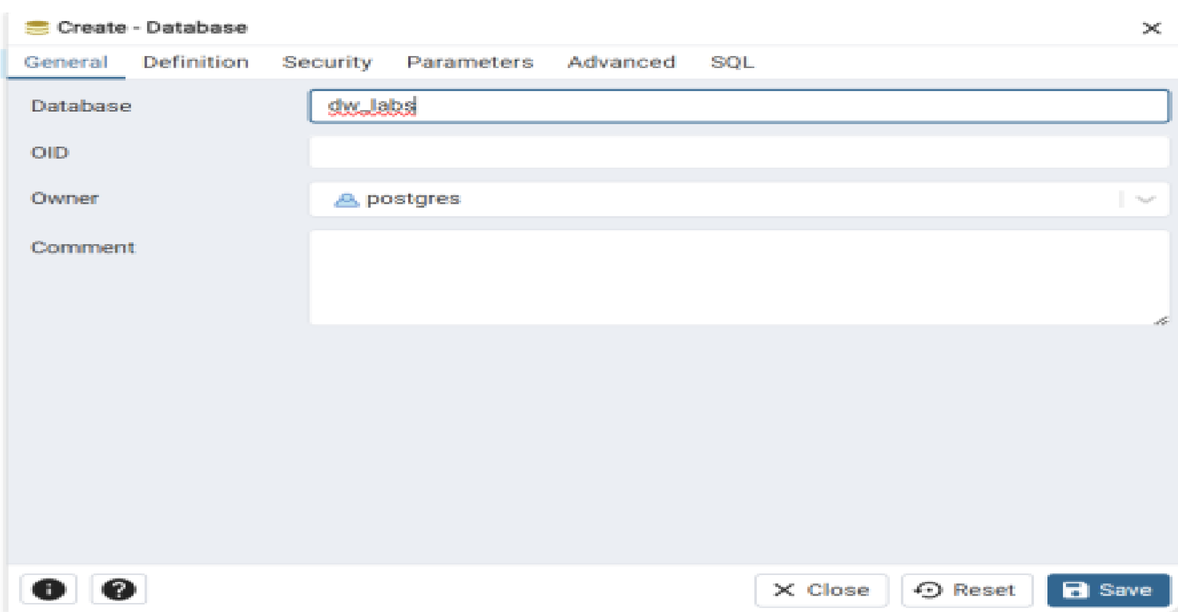


Figure 1: Create Database

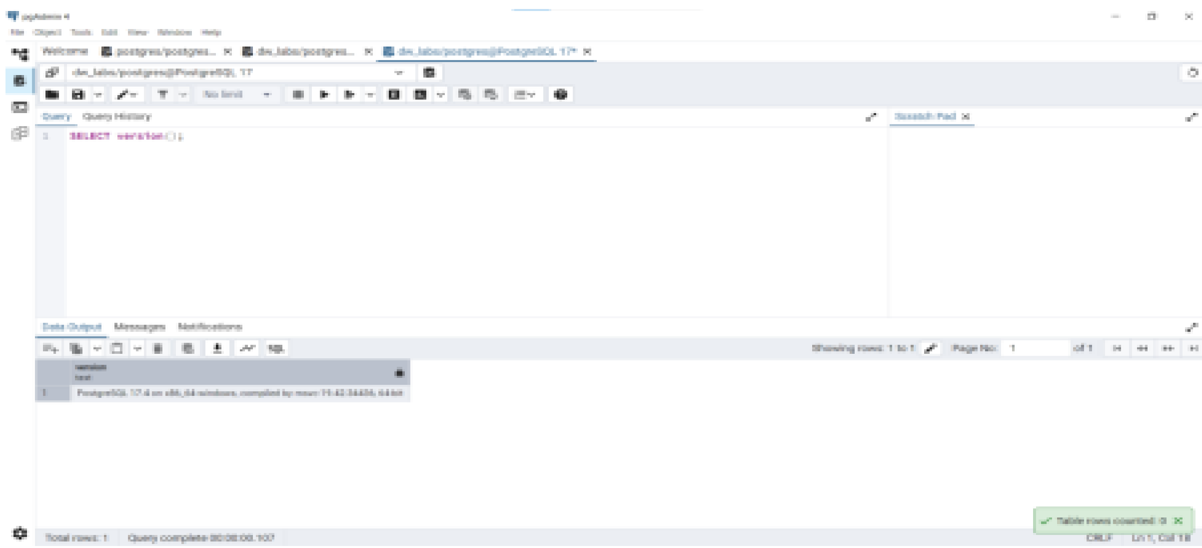


Figure 2: Query Tool

Python Verification (using Jupyter Notebook/Lab):

1. Launch Jupyter Notebook/JupyterLab from Anaconda Navigator or the prompt (jupyter notebook or jupyter lab) to run Python code interactively.
2. Create a new Python 3 no ittebook.
3. In the first cell, type the following code to import libraries and print their versions:

```
import pandas as pd
import numpy as np
import sklearn
import matplotlib
import seaborn as sns
import mlxtend

print(f"Pandas version: {pd.__version__}")
print(f"NumPy version: {np.__version__}")
print(f"Scikit-learn version: {sklearn.__version__}")
print(f"Matplotlib version: {matplotlib.__version__}")
print(f"Seaborn version: {sns.__version__}")
print(f"Mlxtend version: {mlxtend.__version__}")
```

4. Run the cell (Shift+Enter). Verify that the versions are printed without errors.

Exercises

1. Using the pgAdmin Query Tool connected to your dw_labs database, create a simple table named course_info with columns course_id (VARCHAR(10), primary key) and course_name (VARCHAR(100)). Write the SQL command used.

```
CREATE TABLE course_info (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100)
);
```

2. Insert a record into course_info with course_id='CS444' and course_name='Data Warehousing and Mining'. Write the SQL command used.

```
CREATE TABLE course_info (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100)
);
INSERT INTO course_info (course_id, course_name)
VALUES ('CS444', 'Data Warehousing and Mining');
```

3. Write an SQL command to select all data from the course_info table and verify the inserted record.

```
CREATE TABLE course_info (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100)
);
INSERT INTO course_info (course_id, course_name)
VALUES ('CS444', 'Data Warehousing and Mining');
SELECT * FROM course_info;
```

Output:

Course_info

course_id	course_name
CS444	Data Warehousing and Mining

4. In a Jupyter Notebook cell, create a Pandas Series containing the names of the key libraries used in this course (pandas, numpy, sklearn, matplotlib, seaborn, mlxtend). Print the Series.

```
import pandas as pd
libraries = pd.Series(['pandas', 'numpy', 'sklearn', 'matplotlib', 'seaborn',
                       'mlxtend'])
print("Key libraries used in the course:")
print(libraries)
```

Output:

```
Key libraries used in the course:
0        pandas
1         numpy
2       sklearn
3    matplotlib
4       seaborn
5       mlxtend
dtype: object
```

Lab Session 02

Exploring Data Sources & Basic SQL for ETL

Objective

Understand different data sources. Practice using basic SQL commands (SELECT, WHERE, JOIN, GROUP BY, Aggregate functions) crucial for data extraction and understanding.

Dataset

SalesData_OLTP.csv: TransactionID, Timestamp, CustomerID, ProductID, Quantity, UnitPrice, StoreID CustomerInfo.csv: CustomerID, CustomerName, City, State ProductInfo.csv: ProductID, ProductName, Category, SupplierID

Tasks

• **Loading Data from CSV:**

A CSV (Comma-Separated Values) file is a plain text file that stores tabular data, where each line represents a record and each field within a record is separated by commas. These files are widely used for exchanging data between different software applications and are commonly associated with spreadsheets and databases. **Key characteristics of CSV files:**

- o **Plain text:** CSV files are simple text files, making them easily readable and editable with any text editor.
- o **Tabular data:** They store data in a table-like format, where rows represent records and columns represent fields.
- o **Comma delimiter:** Each field within a record is separated by a comma.
- o **Line breaks:** Each new record is typically separated by a line break.
- o **Versatile:** CSV files are widely used for data exchange between various applications, including spreadsheets (like Microsoft Excel), databases, and many other software programs.

• **Method 1: Using pgAdmin Import Tool:**

1. First, create the target tables in PostgreSQL with appropriate data types matching the CSV columns. Example for customer_info:

```
CREATE TABLE customer_info (
 CustomerID VARCHAR(10) PRIMARY KEY,
 CustomerName VARCHAR(100),
 City VARCHAR(50),
 State VARCHAR(50)
);
```

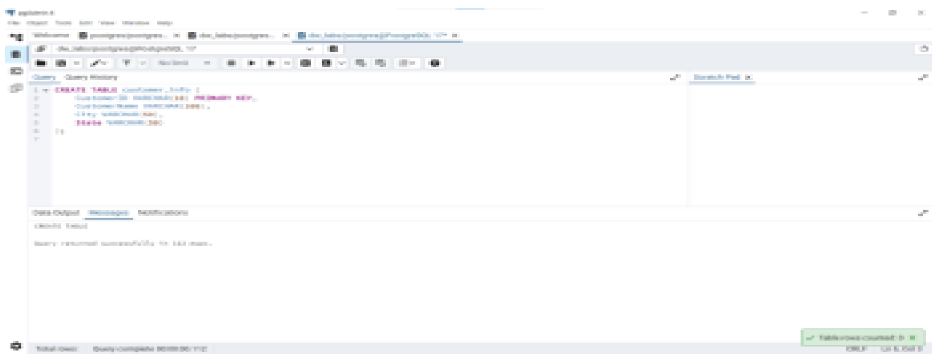


Figure 1: Query Tool

2. In pgAdmin's object browser, right-click the created table (e.g., customer_info). 3. Select 'Import/Export...'.

4. Toggle to 'Import'. Provide the path to your CustomerInfo.csv file. 5. Go to the 'Options' tab. Ensure 'Header' is turned ON if your CSV has a header row. Set the 'Delimiter' (usually comma ',').

6. Click 'OK'. Repeat for ProductInfo.csv and SalesData_OLTP.csv

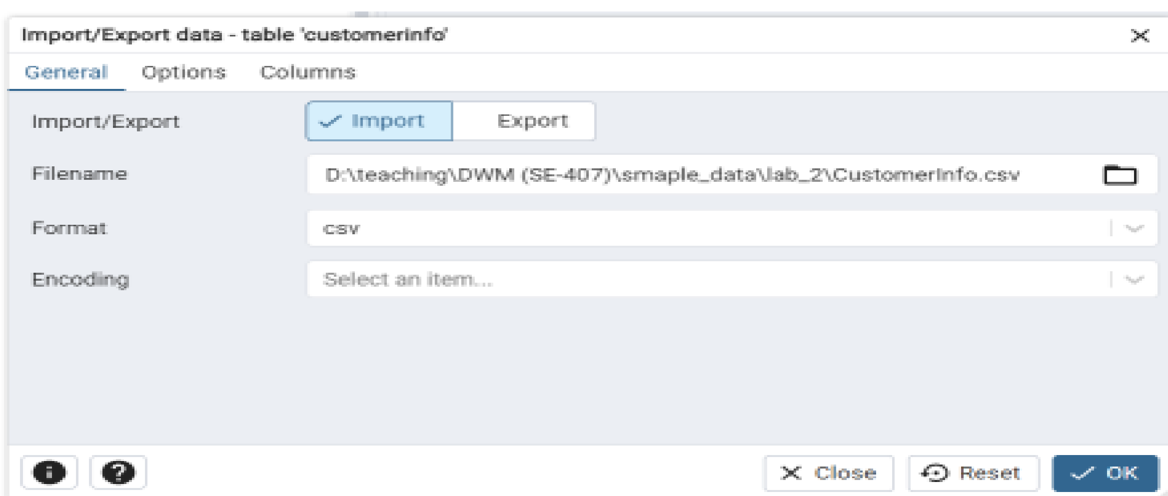
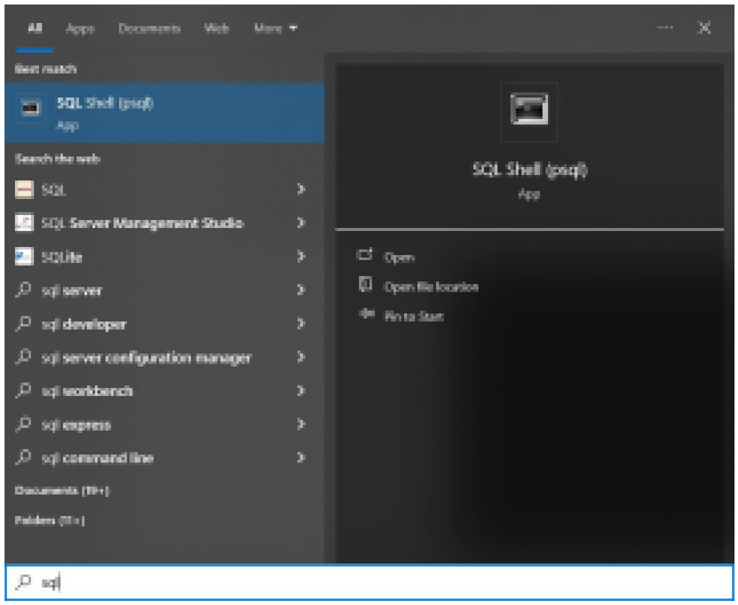


Figure 2: Import/Export utility tool

- **Method 2: Using SQL COPY Command (More powerful):**

1. Open SQL Shell (psql).



2. Run the following command

\copy customer_info FROM '/path/to/your/CustomerInfo.csv' WITH (FORMAT CSV, HEADER); 3. Repeat for other tables/files.

Imagine a scenario in which you have a directory, A, that contains another directory, B. Directory B is then a subdirectory of directory A, and directory A is the parent directory of directory B.

Basic SQL Exploration:

SELECT <columns> FROM <table>: Retrieve specific columns. Use * for all.
WHERE <condition>: Filter rows (e.g., WHERE State = 'California', WHERE Quantity > 10). JOIN: Combine rows from two or more tables based on a related column.
INNER JOIN: Returns only matching rows from both tables.
LEFT JOIN: Returns all rows from the left table, and matched rows from the right1 (or NULLs if no match).

```
SELECT s.*, c.CustomerName -- Select columns from both tables
FROM salesdata_oltp s
INNER JOIN customerinfo c ON s.CustomerID = c.CustomerID; -- Join condition
```

GROUP BY <columns>: Groups rows with the same values in specified columns into a

summary row. Often used with aggregate functions.
Aggregate Functions: COUNT(), SUM(), AVG(), MIN(), MAX().

Applied to groups. SELECT Category, AVG(s.Quantity * s.UnitPrice) AS

AverageSale

FROM salesdata_oltp s
JOIN productinfo p ON s.ProductID =
p.ProductID GROUP BY p.Category; --
Calculate average sale per

Exercise

1. Create the tables SalesData_OLTP, CustomerInfo, and ProductInfo in your dw_labs database with appropriate data types for the columns listed in the Dataset description.

Query Query History

```sql
-- Lab 2: Question 1
CREATE TABLE CustomerInfo (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(255) NOT NULL,
    City VARCHAR(100),
    "State" VARCHAR(50)
);
CREATE TABLE ProductInfo (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255) NOT NULL,
    Category VARCHAR(100),
    SupplierID INT
);
CREATE TABLE SalesData_OLTP(
    TransactionID INT PRIMARY KEY,
    Timestamp TIMESTAMP NOT NULL,
    CustomerID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    UnitPrice DECIMAL(10, 2) NOT NULL CHECK (UnitPrice >= 0),
    StoreID INT NOT NULL,
    FOREIGN KEY(CustomerID) REFERENCES CustomerInfo(customerid),
    FOREIGN KEY(ProductID) REFERENCES ProductInfo(productid)
);
```

2. Load the data from the provided CSV files into these tables using either pgAdmin's Import tool or the \copy command. Verify data is loaded using SELECT COUNT(*) FROM <table_name>; for each table. **3.** Write an SQL query to find the total number of unique customers who made purchases (CustomerID from salesdata_oltp).

```sql
-- Lab 2: Question 2
SELECT * FROM CustomerInfo;
```

Data Output Messages Notifications

	customerid [PK] integer	customername character varying (255)	city character varying (100)	State character varying (50)
1	1	Alice Wonderland	New York	NY
2	2	Bob The Builder	Los Angeles	CA
3	3	Charlie Chaplin	Chicago	IL
4	4	Diana Prince	Boston	MA

```sql
SELECT * FROM ProductInfo;
```

Data Output Messages Notifications

	productid [PK] integer	productname character varying (255)	category character varying (100)	supplierid integer
1	101	Laptop Pro 15	Electronics	501
2	102	Wireless Optical Mouse	Electronics	501
3	103	Mechanical Keyboard RGB	Electronics	502
4	104	Standard Coffee Mug, White	Kitchenware	503
5	105	Office Chair Ergonomic	Furniture	504
6	106	Notebook Spiral A4	Office Supplies	505

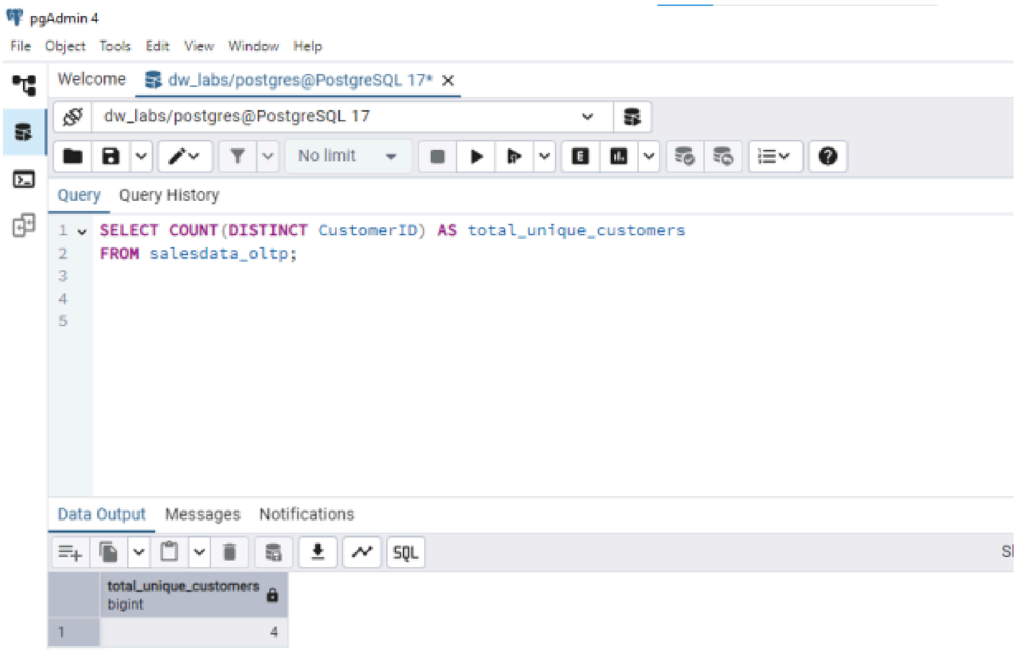
```sql
SELECT * FROM SalesData_OLTP;
```

Data Output Messages Notifications

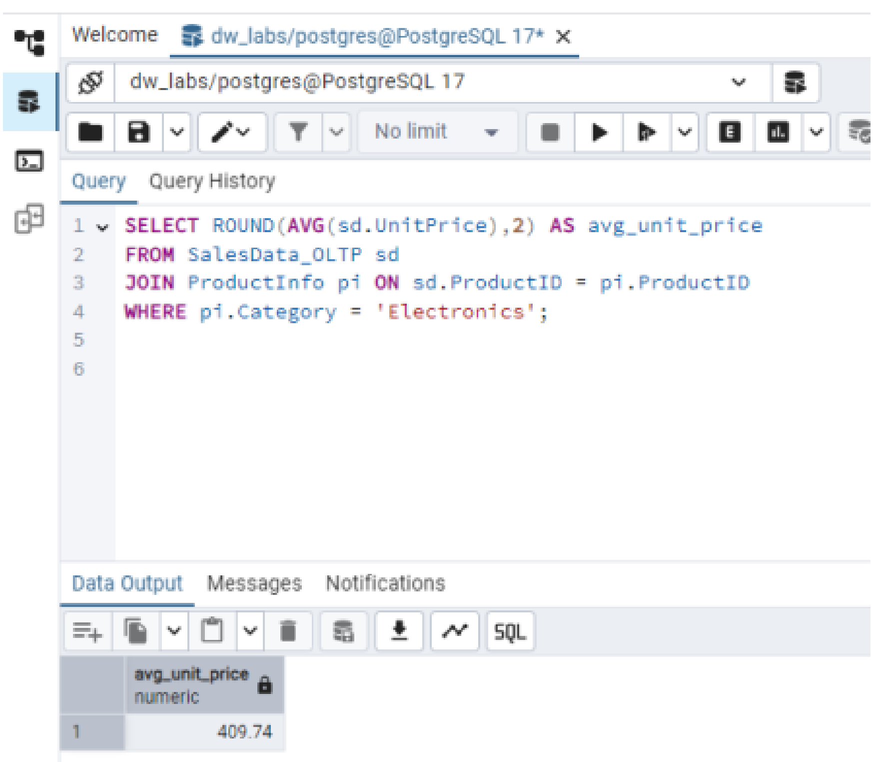
Showi

	transactionid [PK] integer	timestamp timestamp without time zone	customerid integer	productid integer	quantity integer	unitprice numeric (10,2)	storeid integer
1	10001	2025-04-24 10:15:00	1	101	1	1499.99	1
2	10002	2025-04-24 10:15:00	1	102	1	24.50	1
3	10003	2025-04-24 14:05:00	2	105	1	299.00	2
4	10004	2025-04-25 08:30:00	3	103	1	89.95	1
5	10005	2025-04-25 09:10:00	1	104	2	8.50	1
6	10006	2025-04-25 09:12:00	4	106	5	3.99	2
7	10007	2025-04-25 09:14:00	2	102	1	24.50	2

3. Write an SQL query to find the total number of unique customers who made purchases (CustomerID from salesdata_oltp).



4. Write an SQL query to find the average UnitPrice for products in the 'Electronics' Category. (Requires joining salesdata_oltp and productinfo).



5. Write an SQL query to list the CustomerName and the total Quantity of items they purchased, but only for customers in 'New York'(NY). Order the results by total quantity descending. (Requires joining salesdata_oltp and customerinfo, filtering, grouping, and ordering).

Query | Query History

```sql
SELECT ci.CustomerName, SUM(sd.Quantity) AS total_quantity
FROM SalesData_OLTP sd
JOIN customer_info ci ON sd.CustomerID = ci.CustomerID
WHERE ci.city = 'New York'
GROUP BY ci.CustomerName
ORDER BY total_quantity DESC;

```

Data Output | Messages | Notifications

	customername character varying (100)	total_quantity bigint
1	Alice Wonderland	4

Lab Session 03

Designing a Data Warehouse: Star Schema

Objective

Understand and design a Star Schema for a given business process. Identify facts, measures, dimensions, and attributes.

Scenario

Design a data warehouse for the SalesData_OLTP analyzed in Lab 2. The goal is to analyze sales quantity and total sales amount by Product, Customer (City/State), and Time (Day/Month/Year).

Introduction to Dimensional Modeling

Dimensional Modeling is a logical design technique widely used for data warehousing and business intelligence (BI). Its primary goal is to structure data in a way that is intuitive for business users and optimized for high performance querying and analysis. Developed primarily by Ralph Kimball, it contrasts with the highly normalized structures (like 3rd

Normal Form - 3NF) often found in transactional (OLTP) databases.

Core Purpose

The main objectives of dimensional modeling are:

1. **Understandability**: To present data in a clear, simple format that aligns with how business users think about their processes and metrics.
2. **Query Performance:** To optimize the database structure for fast retrieval of data, especially for analytical queries that often involve aggregations (SUM, AVG, COUNT) across large datasets.
3. **Simplicity:** To simplify the complex relationships found in operational systems into a more manageable framework.

Key Components

Dimensional models primarily consist of two types of tables:

Fact Tables:

- **Purpose**: Located at the center of the model, fact tables store the measurements or metrics resulting from business processes or events. These are typically numeric and additive (e.g., Sales Amount, Quantity Sold, Cost, Duration).
- **Content**: They contain:
  - **Numeric Facts/Measures:** The quantitative data being tracked.
  - **Foreign Keys:** Keys that connect to the primary keys of the associated dimension tables. These foreign keys collectively often form the primary key of the fact table.
- **Granularity**: Each row in a fact table corresponds to a specific event or transaction at a defined level of detail (the "grain"). For example, the grain could be "one line item on a sales order" or "a daily summary of website visits".

Dimension Tables:

- **Purpose**: These tables surround the fact table and provide the descriptive context for the facts. They answer the "who, what, where, when, why, and how" related to the business event.

- **Content**: They contain:
  - **Primary Key**: A unique identifier for each dimension record (often a surrogate key - a simple, system-generated integer key).
  - **Descriptive Attributes**: Textual or categorical information that describes the dimension (e.g., Customer Name, Product Category, Store City, Calendar Date). These attributes are used for filtering, grouping, and labeling query results.

Examples: Common dimensions include Time (Date), Product, Customer, Geography (Store, Region), Employee, etc. Attributes within these dimensions provide the details (e.g., DimDate might have attributes like Year, Quarter, Month, DayOfWeek;

DimProduct might have Brand, Category, Size, Color).

Star Schema:
• The most common and simplest form.
• Consists of a central fact table directly linked to several dimension tables.
• The dimension tables are generally denormalized (meaning they aren't broken down into further tables based on normalization rules).
• Looks like a star, with the fact table at the center and dimensions radiating outwards.
• **Advantages**: Simple structure, easy for users to understand, efficient query performance due to fewer joins.

Figure 1: Star Schema

Exercises

1) Identify the main business process for the scenario.

Main Business Process:

Analyzing **sales transactions** (quantity and amount) by **Product**, **Customer (City/State)**, and **Time (Day/Month/Year)**.

2) Identify the facts and their types (additive, semi-additive, non-additive). Define the grain of the fact table.

Fact	Type	Description
QuantitySold	Additive	Can be summed across all dimensions
TotalSalesAmount	Additive	Quantity × Unit Price; can be summed as well

Grain of the Fact Table:
One row per **individual sales transaction line item** (based on TransactionID).

3) For each dimension, list its relevant attributes based on the OLTP source data and analytical needs (e.g., For Product: ProductName, Category; For Customer: CustomerName, City, State; For Time: Date, DayOfWeek, Month, Quarter, Year).

DimProduct

Attribute	Description
ProductID (PK)	Surrogate key
ProductName	Product name
Category	Product category
SupplierID	Supplier ID
SupplierName	Supplier name
SupplierCountry	Country of the supplier

DimCustomer

Attribute	Description
CustomerID (PK)	Surrogate key
CustomerName	Name of the customer
City	City
State	State

DimTime

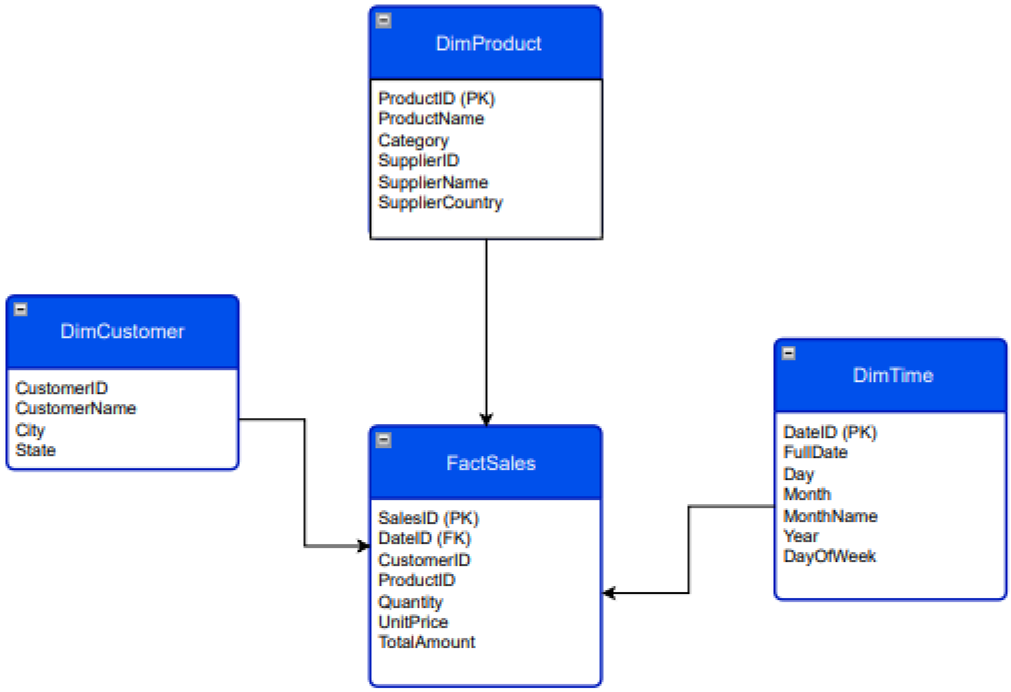
Attribute	Description
DateID (PK)	Surrogate key (Date)

FullDate	Actual Date
Day	Day of the month
Month	Month number
MonthName	Month name (April, etc.)
Year	Year (e.g., 2025)
DayOfWeek	Name of day (e.g., Monday)

FactSales

Attribute	Description
SalesID (PK)	Surrogate key
DateID (FK)	Links to DimTime
CustomerID (FK)	Links to DimCustomer
ProductID (FK)	Links to DimProduct
QuantitySold	Quantity of items sold
UnitPrice	Price per unit
TotalSalesAmount	Quantity × Unit Price

4) Draw the Star Schema diagram, including table names, column names, primary keys (PK), foreign keys (FK), and relationships. Include surrogate keys for dimensions.



5) Write the SQL CREATE TABLE statements for the fact and dimension tables based on your schema design (use appropriate data types).

```sql
CREATE TABLE DimCustomer (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(255),
    City VARCHAR(100),
    State VARCHAR(100)
);

CREATE TABLE DimProduct (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(255),
    Category VARCHAR(100),
    SupplierID INT,
    SupplierName VARCHAR(255),
    SupplierCountry VARCHAR(100)
);
```

```sql
CREATE TABLE DimTime (
    DateID INT PRIMARY KEY,
    FullDate DATE NOT NULL,
    Day INT,
    Month INT,
    MonthName VARCHAR(50),
    Year INT,
    DayOfWeek VARCHAR(50)
);
```

```sql
CREATE TABLE FactSales (
    SalesID INTEGER PRIMARY KEY AUTOINCREMENT,
    DateID INT,
    CustomerID INT,
    ProductID INT,
    Quantity INT,
    UnitPrice DECIMAL(10,2),
    TotalSalesAmount DECIMAL(12,2),

    FOREIGN KEY (DateID) REFERENCES DimTime(DateID),
    FOREIGN KEY (CustomerID) REFERENCES DimCustomer(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID)
);
```

Lab Session 04

Designing a Data Warehouse: Snowflake Schema

Objective

Understand and design a Snowflake Schema. Compare and contrast Star and

Snowflake schemas. **Scenario**

Extend the Sales Data Warehouse design from Lab 3. Assume the DimProduct needs to be normalized further: Category information (e.g., CategoryID, CategoryName) and Supplier information (SupplierID, SupplierName, SupplierCountry) should reside in their own tables, linked from DimProduct.

Snowflake Schema

An extension of the star schema where the dimension tables are normalized into multiple related tables. For example, a DimProduct dimension might be snowflaked into DimProduct, DimCategory, and DimBrand tables. Looks like a snowflake, with branches extending off the main dimensions.
Advantages: Reduces data redundancy, potentially saves storage space.
Disadvantages: More complex structure, requires more joins for queries which can impact performance and user understanding.

Decision criteria for choosing between Star and Snowflake

The choice between a Star schema and a Snowflake schema in data warehousing depends on the specific needs of the data and the reporting requirements. A Star schema is simpler, faster for querying, and easier to understand, making it suitable for many business intelligence scenarios. A Snowflake schema, on the other hand, reduces data redundancy, saves storage space, and maintains data integrity, but can result in slower query times and more complex query design.

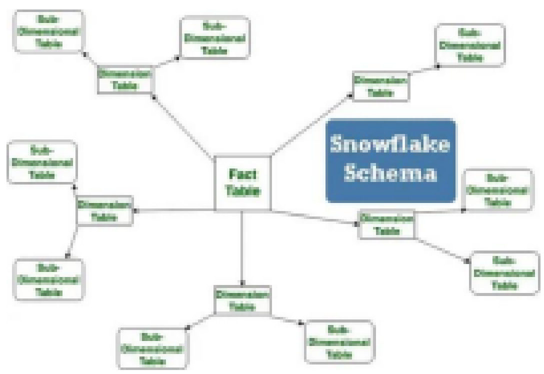


Figure 1: Snowflake Schema

Exercises

1. Based on the Lab 3 Star Schema and the new requirement, identify which dimension(s) should be normalized (snowflaked).

Due to updated ProductInfo table (with Category, SupplierName, and SupplierCountry), the dimension that needs to be snowflaked is: DimProduct because: • Category is repeating for many products so they should go to a separate DimCategory • SupplierID, SupplierName, and SupplierCountry repeat so they should go to a separate DimSupplier Q

2. Design the new dimension tables required for the snowflaking (e.g., DimCategory, DimSupplier). Define their attributes and primary keys.

a) **DimCategory:**

Column Name	Data Type	Description
CategoryID	INT (PK)	Surrogate key
Category	VARCHAR (100)	Category name (eg, Electronics)

b) **DimSupplier:**

Column Name	Data Type	Description
SupplierID	INT (PK)	Unique supplier ID
SupplierName	VARCHAR (100)	Name of supplier
SupplierCountry	VARCHAR (100)	Country of supplier

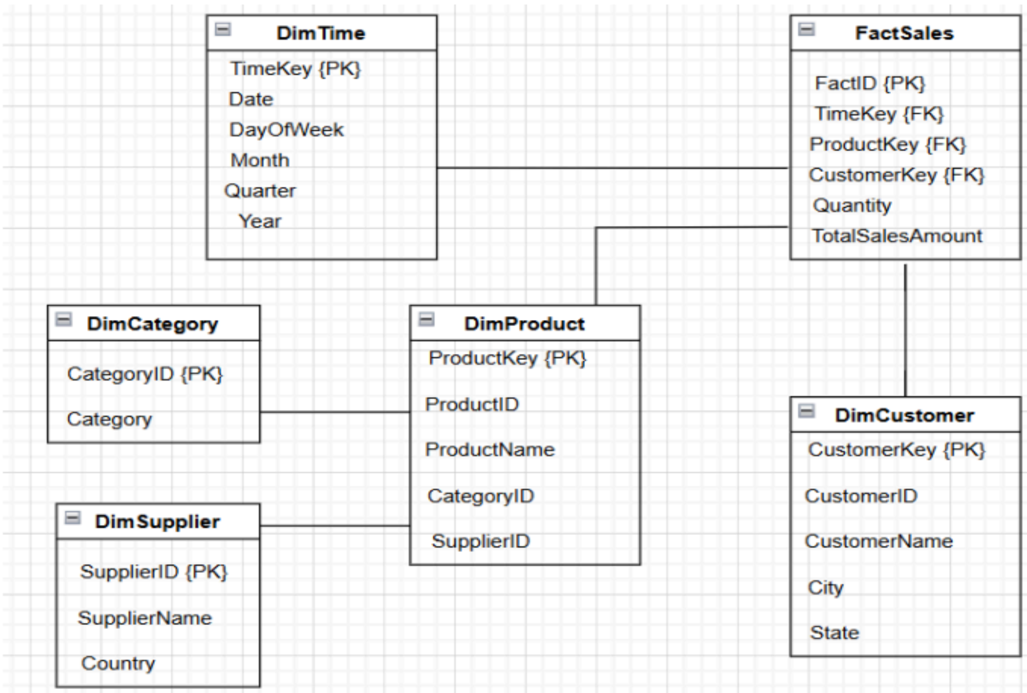
3. Redesign the DimProduct table to link to these new tables using foreign keys, removing the redundant attributes.

Now, DimProduct will no longer store Category, SupplierName, or SupplierCountry directly. Instead, it will link to:

- DimCategory via CategoryID
- DimSupplier via SupplierID

Column Name	Data Type
ProductKey	INT (PK)
ProductID	INT
ProductName	VARCHAR (255)
CategoryID	INT (FK)
SupplierID	INT (FK)

4. Draw the resulting Snowflake Schema diagram, showing the relationships between FactSales, the main dimension tables, and the snowflaked dimension tables.



5. Write the SQL CREATE TABLE statements for the new or modified dimension tables (DimCategory, DimSupplier, updated DimProduct)

Query   Query History

```sql
CREATE TABLE DimCategory (
    CategoryID SERIAL PRIMARY KEY,
    Category VARCHAR(100) NOT NULL
);
CREATE TABLE DimSupplier (
    SupplierID INT PRIMARY KEY,
    SupplierName VARCHAR(255) NOT NULL,
    SupplierCountry VARCHAR(100)
);
CREATE TABLE Dim_Product (
    ProductKey SERIAL PRIMARY KEY,
    ProductID INT NOT NULL,
    ProductName VARCHAR(255) NOT NULL,
    CategoryID INT,
    SupplierID INT,
    FOREIGN KEY (CategoryID) REFERENCES DimCategory(CategoryID),
    FOREIGN KEY (SupplierID) REFERENCES DimSupplier(SupplierID)
);
```

Lab Session 05

Implementing ETL - Extraction & Transformation

Objective

Implement the 'E' (Extract) and 'T' (Transform) phases of the ETL process using SQL. Handle data type conversions, basic cleaning, and lookups.

Scenario

Populate the Star Schema DimProduct and DimCustomer tables designed in Lab 3 from the product_info and customer_info source tables created in Lab 2.

ETL (Extract, Transform, Load) Overview

ETL stands for Extract, Transform, and Load. It's a crucial process used to collect data from various sources, convert it into a consistent and usable format, and store it in a target system, typically a data warehouse (DW), data mart, or data lake. The goal is to prepare data for analysis, reporting, and business intelligence.

1. **Extract**
   - **Goal**: To retrieve raw data from one or more source systems.
   - **Sources**: Data can come from a wide variety of places, including:
     - o Relational Databases (like PostgreSQL, MySQL, SQL Server, Oracle)
     - o Flat Files (CSV, JSON, XML, Log files)
     - o APIs (Web services, social media platforms)
     - o NoSQL Databases
     - o Spreadsheets
     - o Legacy Systems
     - o Cloud Storage
   - **Process**: This stage involves connecting to the source systems and extracting the required data. Challenges often include handling different data formats, dealing with source system availability, and minimizing the impact on the performance of operational systems (often done during off-peak hours). Data extraction can be full (all data) or incremental (only changes since the last extraction).

2. **Transform**
   - **Goal**: To cleanse, reshape, and enrich the extracted raw data to make it consistent, accurate, and suitable for analysis in the target system. This is often the most complex stage.

   - **Common Operations**:
     - **Cleaning**: Handling missing values (imputation or removal), correcting errors (e.g.,

typos in state names), standardizing formats (e.g., date formats, units of measure).
- **Validation**: Checking if data conforms to predefined rules or constraints (e.g., ensuring product codes exist in the product dimension).
- **Deduplication**: Identifying and removing duplicate records.
- **Integration**: Combining data from multiple sources (e.g., merging customer data from sales and marketing systems).

- **Enrichment**: Adding calculated values (e.g., calculating TotalSaleAmount from Quantity * UnitPrice), deriving new attributes (e.g., extracting month from a date), or performing lookups (e.g., finding a DimCustomerID surrogate key based on the source CustomerID).
- **Restructuring/Reshaping**: Pivoting or unpivoting data, splitting or merging columns, converting data types to match the target schema (e.g., the star schema).
- **Aggregation**: Summarizing data to a higher level (e.g., calculating daily totals from individual transactions), although this is sometimes done during the load phase or within the BI tool itself.

- **Process**: Transformations are applied based on business rules and the requirements of the target data model (like your star schema). This often involves staging areas where data is temporarily held during transformation.

3. Load

- **Goal:** To write the transformed data into the final target system (the data warehouse or data mart).

- **Target:** Typically, a relational database optimized for analytical queries (like the star schema tables we designed: FactSales, DimCustomer, DimProduct, etc.).

- **Loading Strategies:**
  - o **Full Load:** Erasing the existing data in the target table(s) and loading all the transformed data. Simpler but potentially slow for large datasets.
  - o **Incremental Load (Delta Load):** Loading only the new or changed data since the last load cycle. More complex to implement (requires tracking changes) but much faster and more efficient for ongoing updates. This often involves techniques like Change Data Capture (CDC).
  - o **Update/Upsert:** Updating existing records in the target if they've changed in the source, and inserting new records. Crucial for managing dimensions (especially Slowly Changing Dimensions - SCDs).

- **Process:** This stage involves writing the prepared data into the dimension and fact tables, ensuring referential integrity (making sure foreign keys in the fact table correctly reference primary keys in the dimension tables), and potentially building indexes or performing other post-load optimizations.

Extraction

Simple SELECT statements from source tables (customer_info, product_info).

Transformation using SQL Functions

• Handling NULLs: COALESCE(column_name, 'default_value') replaces NULL with a default. • Changing Case: UPPER(column_name), LOWER(column_name).

• Trimming Spaces: TRIM(column_name) removes leading/trailing whitespace.

• Data Type Casting: CAST(column_name AS new_type) or column_name::new_type (PostgreSQL shorthand). Example: some_text_column::INT.

• String Concatenation: column1 || ' ' || column2.

Loading Dimensions

Use INSERT INTO ... SELECT ... statement. The SELECT part extracts and transforms data from the source, and the INSERT INTO part loads it into the target dimension table. Surrogate keys (ProductKey, CustomerKey) will be generated automatically by the SERIAL definition.

```
-- Example: Populate DimCustomer from customerinfo
INSERT INTO DimCustomer (CustomerID, CustomerName, City, State)
SELECT
CustomerID, -- Source CustomerID
UPPER(TRIM(CustomerName)), -- Transform: Trim spaces, convert to uppercase
COALESCE(City, 'Unknown'), -- Transform: Handle NULL City
State -- Source State (assuming no transform needed)
FROM customerinfo; -- Extract from source
```

Staging Area (Concept)

In real-world scenarios, data is often extracted to intermediate 'staging' tables before transformation and loading into final DW tables. This isolates processes and aids troubleshooting. We are skipping explicit staging tables here for simplicity.

Exercise

1. Ensure your Star Schema dimension tables (DimProduct, DimCustomer from Lab 3) exist and are empty. (You might need TRUNCATE TABLE <table_name>; if they have data from previous runs). Ensure source tables (product_info, customer_info from Lab 2) are populated.

CREATION OF TABLES :

```sql
CREATE TABLE DimProduct (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    category_id INT
);
```

```sql
CREATE TABLE DimCustomer (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    city VARCHAR(50),
    state CHAR(2)
);
```

```sql
CREATE TABLE Time_Dim (
    time_id INT PRIMARY KEY,
    full_date DATE,
    day INT,
    month INT,
    year INT
);
```

```sql
CREATE TABLE Sales_Fact (
    sale_id INT PRIMARY KEY,
    time_id INT,
    customer_id INT,
    product_id INT,
    quantity INT,
    price DECIMAL(10,2),
    FOREIGN KEY (time_id) REFERENCES Time_Dim(time_id),
    FOREIGN KEY (customer_id) REFERENCES DimCustomer(customer_id),
    FOREIGN KEY (product_id) REFERENCES DimProduct(product_id)
);
```

INSERTION OF VALUES:

```sql
INSERT INTO DimProduct (product_id, product_name, category, category_id) VALUES
(101, 'Laptop Pro 15', 'Electronics', 501),
(102, 'Wireless Optical Mouse', 'Electronics', 501),
(103, 'Mechanical Keyboard RGB', 'Electronics', 502),
(104, 'Standard Coffee Mug, White', 'Kitchenware', 503),
(105, 'Office Chair Ergonomic', 'Furniture', 504),
(106, 'Notebook Spiral A4', 'Office Supplies', 505);
```

```sql
INSERT INTO DimCustomer (customer_id, customer_name, city, state) VALUES
(1, 'Alice Wonderland', 'New York', 'NY'),
(2, 'Bob The Builder', 'Los Angeles', 'CA'),
(3, 'Charlie Chaplin', 'Chicago', 'IL'),
(4, 'Diana Prince', 'Boston', 'MA');
```

```sql
INSERT INTO Time_Dim (time_id, full_date, day, month, year) VALUES
(1, '2025-04-24', 24, 4, 2025),
(2, '2025-04-25', 25, 4, 2025);
```

```sql
INSERT INTO Sales_Fact (sale_id, time_id, customer_id, product_id, quantity, price) VALUES
(10001, 1, 1, 101, 1, 1499.99),
(10002, 1, 1, 102, 1, 24.50),
(10003, 2, 2, 105, 1, 299.00),
(10004, 2, 3, 103, 1, 89.95),
(10005, 2, 1, 104, 2, 8.50),
(10006, 2, 4, 106, 5, 3.99),
(10007, 2, 2, 102, 1, 24.50);
```

TEST QUERY:

	product_id [PK] integer	product_name character varying (100)	category character varying (50)	category_id integer
1	101	Laptop Pro 15	Electronics	501
2	102	Wireless Optical Mouse	Electronics	501
3	103	Mechanical Keyboard RGB	Electronics	502
4	104	Standard Coffee Mug, White	Kitchenware	503
5	105	Office Chair Ergonomic	Furniture	504
6	106	Notebook Spiral A4	Office Supplies	505

	customer_id [PK] integer	customer_name character varying (100)	city character varying (50)	state character (2)
1	1	Alice Wonderland	New York	NY
2	2	Bob The Builder	Los Angeles	CA
3	3	Charlie Chaplin	Chicago	IL
4	4	Diana Prince	Boston	MA

2. Write and execute a INSERT INTO ... SELECT ... statement to populate DimProduct from productinfo. • Select ProductID, ProductName, Category, SupplierID.
• Handle potential NULL values in Category by replacing them with 'N/A'.
• Ensure ProductName has leading/trailing whitespace removed using TRIM().

```
INSERT INTO DimProduct (product_id, product_name, category, category_id)
SELECT
    ProductID,
    TRIM(ProductName),
    COALESCE(Category, 'N/A'),
    SupplierID
FROM product_info
WHERE ProductID NOT IN (SELECT product_id FROM DimProduct);
```

3. Write and execute a INSERT INTO ... SELECT ... statement to populate DimCustomer from customer_info. • Select CustomerID, CustomerName, City, State.
• Convert CustomerName to uppercase using UPPER().
• Replace NULL values in City with 'Unknown'.

```
INSERT INTO DimCustomer (customer_id, customer_name, city, state)
SELECT
    CustomerID,
    UPPER(CustomerName),
    COALESCE(City, 'Unknown'),
    State
FROM customer_info
WHERE CustomerID NOT IN (SELECT customer_id FROM DimCustomer);
```

4. Verify the data loaded into DimProduct and DimCustomer using SELECT * FROM <table_name> LIMIT 10; . Check if surrogate keys were generated and transformations applied.

```
SELECT * FROM DimProduct LIMIT 10;
```

	product_id [PK] integer	product_name character varying (100)	category character varying (50)	category_id integer
1	101	Laptop Pro 15	Electronics	501
2	102	Wireless Optical Mouse	Electronics	501
3	103	Mechanical Keyboard RGB	Electronics	502
4	104	Standard Coffee Mug, White	Kitchenware	503
5	105	Office Chair Ergonomic	Furniture	504
6	106	Notebook Spiral A4	Office Supplies	505

```sql
SELECT * FROM DimCustomer LIMIT 10;
```

	customer_id [PK] integer	customer_name character varying (100)	city character varying (50)	state character (2)
1	1	Alice Wonderland	New York	NY
2	2	Bob The Builder	Los Angeles	CA
3	3	Charlie Chaplin	Chicago	IL
4	4	Diana Prince	Boston	MA
5	5	JOHN DOE	Unknown	TX

Lab Session 06

Implementing ETL - Loading & Basic Data Quality

Objective

Implement the 'L' (Load) phase for the fact table. Perform lookups to get surrogate keys.

Implement basic data quality checks.

Scenario
Populate the FactSales table (Star Schema, Lab 3) using data from the salesdata_oltp source table and the dimension tables (DimProduct, DimCustomer, DimTime) populated in Lab 5.

Tasks:

Populating DimTime

Time dimensions are usually not derived entirely from transaction data. They are often pre-populated using scripts or generated procedurally. For this lab, we need a few representative rows.

```
SELECT DISTINCT(DATE(Timestamp)) FROM salesdata_oltp;
```

	date date
1	2025-04-24
2	2025-04-25

```
-- Insert data for April 24, 2025
INSERT INTO DimDate (DateKey, FullDate, DayName, DayOfMonth, Month,
Quarter, Year) VALUES (20250424, '2025-04-24', 'Thursday', 24, 4, 2, 2025);

-- Insert data for April 25, 2025
INSERT INTO DimDate (DateKey, FullDate, DayName, DayOfMonth, Month,
Quarter, Year) VALUES (20250425, '2025-04-25', 'Friday', 25, 4, 2, 2025);
```

Loading the Fact Table (FactSales)

This involves joining the source transaction table (salesdata_oltp) with the dimension tables (DimProduct, DimCustomer, DimTime) on their natural keys (e.g., ProductID, CustomerID, Date) to look up the corresponding surrogate keys (ProductKey, CustomerKey, TimeKey.

Lookup Joins
Use LEFT JOIN from the fact source (salesdata_oltp) to dimensions. LEFT JOIN is safer than INNER JOIN because it preserves all transaction rows even if a corresponding dimension entry is missing (though this indicates a data quality issue).

Handling Missing Lookups

Use COALESCE(DimensionTable.SurrogateKey, -1) to insert a default value (e.g., -1, assuming you have an 'Unknown' record with PK=-1 in your dimensions) if a lookup fails. For simplicity here, we'll start assuming lookups succeed or use INNER JOIN first.

Date/Time Key Lookup

Converting the Timestamp from salesdata_oltp to match the DimTime.TimeKey format is crucial. If TimeKey is YYYYMMDD (integer):

```
--PostgreSQL function to convert timestamp to YYYYMMDD integer
TO_CHAR(s.Timestamp, 'YYYYMMDD')::INT
```

Putting it Together (INSERT INTO FactSales...)

```
INSERT INTO FactSales (ProductKey, CustomerKey, TimeKey, QuantitySold,
TotalSalesAmount) SELECT
 -- Lookups for Surrogate Keys
 dp.ProductKey,
 dc.CustomerKey,
 dt.TimeKey,
 -- Measures from source
 s.Quantity,
 (s.Quantity * s.UnitPrice)-- Calculated measure

FROM salesdata_oltp s
-- Join to dimensions to get surrogate keys
INNER JOIN DimProduct dp ON s.ProductID = CAST(dp.ProductID AS INTEGER)
INNER JOIN DimCustomer dc ON s.CustomerID = CAST(dc.CustomerID
AS INTEGER) INNER JOIN DimTime dt ON TO_CHAR(s.Timestamp,
'YYYYMMDD')::INT = dt.TimeKey; -- Using INNER JOIN assumes all
products/customers/dates in salesdata_oltp exist in dimensions. -- Switch to
LEFT JOIN and COALESCE for robustness later.
```

Basic Data Quality Checks

Row Counts: SELECT COUNT(*) FROM FactSales; vs SELECT COUNT(*) FROM salesdata_oltp; (Should match if using INNER JOIN and all lookups succeed).
Referential Integrity: Check for NULL foreign keys in the fact table (shouldn't happen with INNER JOIN, but possible with LEFT JOIN if COALESCE wasn't used or default key doesn't exist). SELECT COUNT(*) FROM FactSales WHERE ProductKey IS NULL;
Measure Validity: Check for NULLs or unreasonable values in measures. SELECT COUNT(*) FROM FactSales WHERE TotalSalesAmount <= 0;

EXERCISE

1. Manually insert at least 5-10 rows into your DimTime table, ensuring the dates cover the range found in your salesdata_oltp data. Use the YYYYMMDD integer format for TimeKey and derive the other columns (FullDate, Year, etc.).

```sql
SELECT SUM(fs.Quantity) AS TotalQuantitySold_2025
FROM FactSales fs
JOIN DimTime dt ON fs.TimeKey = dt.TimeKey
WHERE dt.Year = 2023;
```

Output Messages Notifications

totalquantitysold_2025 numeric
8.00

2. Write and execute the PostgreSQL INSERT INTO FactSales ... SELECT ... statement as shown in the Lab Content (using INNER JOINs for now) to populate FactSales from salesdata_oltp, looking up surrogate keys from DimProduct, DimCustomer, and DimTime. Remember to calculate TotalSalesAmount.

```sql
SELECT COALESCE(SUM(fs.TotalAmount), 0) AS TotalSalesAmount_LaptopPro_NY
FROM FactSales fs
JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
WHERE dp.ProductName = 'Laptop Pro'
  AND dc.State = 'NY';
```

utput Messages Notifications

totalsalesamount_laptoppro_ny numeric
3000.00

3. Verify that rows were loaded into FactSales using SELECT COUNT(*) FROM FactSales;. Compare this count to SELECT COUNT(*) FROM salesdata_oltp;. Are they the same? Why

or why not?

```sql
SELECT dc.State, cat.CategoryName, SUM(fs.TotalAmount) AS TotalSalesAmount
FROM FactSales fs
JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
JOIN DimCategory cat ON dp.CategoryID = cat.CategoryID
GROUP BY dc.State, cat.CategoryName
ORDER BY dc.State, cat.CategoryName;
```

)utput Messages Notifications

state character varying (50)	categoryname character varying (50)	totalsalesamount numeric
CA	Clothing	300.00
NY	Electronics	4000.00
Sindh	Furniture	500.00

4. Write a query to check if there are any rows in FactSales where the lookup for ProductKey might have conceptually failed (i.e., check for NULLs, although INNER JOIN prevents this). Now, modify the INSERT statement from question 2 to use LEFT JOIN for all dimensions and COALESCE(..., -1) for the keys. Note: You would need to first insert 'Unknown' rows with PK=-1 into your dimension tables for this to work properly. (For this exercise, just write the modified query structure).

```sql
WITH StateSales AS (
    SELECT dc.State, SUM(fs.TotalAmount) AS TotalSales
    FROM FactSales fs
    JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
    GROUP BY dc.State
),
TopState AS (
    SELECT State
    FROM StateSales
    ORDER BY TotalSales DESC
    LIMIT 1
)
SELECT dc.City, SUM(fs.TotalAmount) AS TotalSalesAmount
FROM FactSales fs
JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
WHERE dc.State = (SELECT State FROM TopState)
GROUP BY dc.City;
```

a Output Messages Notifications

city character varying (50)	totalsalesamount numeric
New York	4000.00

Lab Session 07

OLAP Operations using SQL

Objective

Perform Online Analytical Processing (OLAP) operations (Slice, Dice, Roll-up, Drill-down, Pivot) on the data warehouse using standard SQL queries.

Scenario

Use the populated FactSales and Dimension tables (Star Schema from Lab 3) to answer analytical business questions.

OLAP (Online Analytical Processing) Concepts

OLAP is a category of software technology that enables analysts, managers, and executives to gain insights from data through fast, consistent, interactive access to a wide variety of possible views of information. The data is typically sourced from data warehouses (often structured using dimensional models like star schemas) and transformed into a structure optimized for analysis.

The fundamental concept behind OLAP is viewing data in multiple dimensions. Imagine a cube of data where each axis represents a different business dimension (like Time, Product, Geography) and the cells within the cube represent the measures or facts (like Sales Amount, Quantity Sold). OLAP allows users to navigate and analyze this multidimensional data structure intuitively.

Key Characteristics:

- **Multidimensional View**: Presents data logically according to business dimensions, not the underlying physical storage.
- **Fast Performance**: Designed for rapid query response, allowing users to explore data interactively without long waits. Queries typically involve aggregations over large datasets.
- **Interactive**: Users can dynamically manipulate data views, filter, sort, and drill down/up to explore different perspectives.
- **Consistent Reporting**: Ensures that calculations and data definitions are consistent across different views and reports.

Common OLAP Operations:

1. **Slice:** Reduces the dimensionality of the cube by selecting a single value for one dimension (e.g., viewing sales data *only* for the 'Electronics' category). This is like taking a slice out of the cube.

2. **Dice:** Selects specific values for *multiple* dimensions to create a smaller sub-cube for analysis (e.g., viewing sales data for 'Laptops' (Product dimension) in the 'North' region (Geography dimension) during 'Q1' (Time dimension)).

3. **Drill Down:** Navigates from less detailed data to more detailed data within a dimension hierarchy (e.g., moving from 'Year' -> 'Quarter' -> 'Month' in the Time dimension, or from 'Category' -> 'ProductName' in the Product dimension).

4. **Drill Up (or Roll-Up):** The opposite of drill down; aggregates data along a dimension hierarchy, moving from more detailed data to less detailed data (e.g., summing monthly sales figures to get quarterly totals, or moving from 'City' -> 'State' -> 'Country').
5. **Pivot (Rotate):** Rotates the axes of the data cube to provide a different perspective on the data (e.g., swapping Products and Time on the axes of a report to see sales trends per product instead of product sales per time period).

Benefits of OLAP

- Facilitates complex analysis and ad-hoc querying.
- Provides a consistent view of business information.
- Enables faster decision-making through quick data exploration.
- Empowers business users to perform their own analysis without deep technical knowledge of underlying data structures.

OLAP & SQL

Simulating OLAP cube operations using standard SQL on a Star Schema.

Slice: Filtering on a single dimension attribute. Achieved using the WHERE clause on a joined dimension table column.

```
-- Slice: Total sales amount for 'Electronics' category
SELECT SUM(fs.TotalSalesAmount)
FROM FactSales fs
JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
WHERE dp.Category = 'Electronics';
```

Dice: Filtering on multiple dimension attributes across one or more dimensions. Achieved using multiple conditions in the WHERE clause combined with AND.

```
-- Dice: Total quantity sold for 'Electronics' in 'California' during 2025
SELECT SUM(fs.QuantitySold)
FROM FactSales fs
JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
JOIN DimTime dt ON fs.TimeKey = dt.TimeKey
WHERE dp.Category = 'Electronics'
 AND dc.State = 'California'
 AND dt.Year = 2025;
```

Roll-up: Aggregating data to a higher level within a dimension hierarchy (e.g., from monthly sales to yearly sales, or city to state). Achieved using GROUP BY on fewer or higher-level attributes.

```
-- Sales per Category (Lower Level)
```

```
SELECT dp.Category, SUM(fs.TotalSalesAmount) AS TotalSales
FROM FactSales fs JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
GROUP BY dp.Category;

-- Sales Grand Total (Higher Level - Rollup from Category)
SELECT SUM(fs.TotalSalesAmount) AS GrandTotalSales
FROM FactSales fs;
```

Drill-down: Navigating from summarized data to more detail (opposite of Roll-up). Achieved by adding more attributes to GROUP BY or adding finer-grained filters in WHERE.

```
-- Start with Sales per State
SELECT dc.State, SUM(fs.TotalSalesAmount) AS StateSales
FROM FactSales fs JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
GROUP BY dc.State;

-- Drill-down: Show Sales per City within 'California'
SELECT dc.City, SUM(fs.TotalSalesAmount) AS CitySales
FROM FactSales fs JOIN DimCustomer dc ON fs.CustomerKey = dc.CustomerKey
WHERE dc.State = 'California' -- Filter to specific state
GROUP BY dc.City; -- Group by finer attribute (City)
```

Pivot (Conditional Aggregation): Rotating data axes (e.g., years as rows, categories as columns). Standard SQL uses CASE statements inside aggregate functions.

```
-- Pivot: Total Quantity Sold per Year (rows) and Category (columns)
SELECT
 dt.Year,
 SUM(CASE WHEN dp.Category = 'Electronics' THEN fs.QuantitySold ELSE 0 END)
AS  ElectronicsQty,
 SUM(CASE WHEN dp.Category = 'Clothing' THEN fs.QuantitySold ELSE 0 END)
AS ClothingQty,  SUM(CASE WHEN dp.Category = 'Groceries' THEN
fs.QuantitySold ELSE 0 END) AS GroceriesQty  -- Add more CASE statements for
other categories
FROM FactSales fs
JOIN DimTime dt ON fs.TimeKey = dt.TimeKey
JOIN DimProduct dp ON fs.ProductKey = dp.ProductKey
GROUP BY dt.Year
ORDER BY dt.Year;
```

EXERCISES

1) Slice: Write a SQL query to find the total QuantitySold for the Year 2025 only.

```sql
SELECT SUM(sf.quantity) AS TotalQuantitySold
FROM Sales_Fact sf
JOIN Time_Dim td ON sf.time_id = td.time_id
WHERE td.year = 2025;
```

	totalquantitysold bigint
1	7

2) Dice: Write a SQL query to find the total TotalSalesAmount for ProductName = 'Laptop Pro' and Customer State = 'New York'.

```sql
SELECT SUM(sf.quantity * sf.price) AS TotalSalesAmount
FROM Sales_Fact sf
JOIN DimProduct dp ON sf.product_id = dp.product_id
JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
WHERE dp.product_name = 'Laptop Pro 15'
  AND dc.state = 'NY';
```

	totalsalesamount numeric
1	1499.99

3) Roll-up: Write a SQL query to find the total TotalSalesAmount grouped first by State and Category. Then, modify the query (or write a new one) to show the total sales amount grouped only by State (rolling up from Category).

```sql
SELECT dc.state, dp.category, SUM(sf.quantity * sf.price) AS TotalSalesAmount
FROM Sales_Fact sf
JOIN DimProduct dp ON sf.product_id = dp.product_id
JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.state, dp.category
ORDER BY dc.state, dp.category;
```

	state character (2)	category character varying (50)	totalsalesamount numeric
1	CA	Electronics	24.50
2	CA	Furniture	299.00
3	IL	Electronics	89.95
4	MA	Office Supplies	19.95
5	NY	Electronics	1524.49
6	NY	Kitchenware	17.00

```sql
SELECT dc.state, SUM(sf.quantity * sf.price) AS TotalSalesAmount
FROM Sales_Fact sf
JOIN DimProduct dp ON sf.product_id = dp.product_id
JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.state
ORDER BY dc.state;
```

	state character (2)	totalsalesamount numeric
1	CA	323.50
2	IL	89.95
3	MA	19.95
4	NY	1541.49

4) Drill-down: Start with the query showing total sales per State. Write a query to drill down and show the total sales per City but only for the state with the highest total sales (you might need a subquery or Common Table Expression (CTE) to find the top state first).

```sql
SELECT dc.state, SUM(sf.quantity * sf.price) AS TotalSalesAmount
FROM Sales_Fact sf
JOIN DimProduct dp ON sf.product_id = dp.product_id
JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
GROUP BY dc.state
ORDER BY TotalSalesAmount DESC;
```

	state character (2)	totalsalesamount numeric
1	NY	1541.49
2	CA	323.50
3	IL	89.95
4	MA	19.95

```
WITH MaxState AS (
    SELECT dc.state, SUM(sf.quantity * sf.price) AS TotalSalesAmount
    FROM Sales_Fact sf
    JOIN DimProduct dp ON sf.product_id = dp.product_id
    JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
    GROUP BY dc.state
    ORDER BY TotalSalesAmount DESC
    LIMIT 1
)
SELECT dc.city, SUM(sf.quantity * sf.price) AS TotalSalesAmount
FROM Sales_Fact sf
JOIN DimCustomer dc ON sf.customer_id = dc.customer_id
JOIN MaxState ms ON dc.state = ms.state
GROUP BY dc.city
ORDER BY TotalSalesAmount DESC;
```

	city character varying (50)	totalsalesamount numeric
1	New York	1541.49

Lab Session 08

Introduction to Data Mining & Data Preprocessing

Objective
Introduce data mining tasks (classification, clustering, association). Understand the need for data preprocessing. Practice basic data loading and exploration using Python (Pandas).

Dataset
StudentPerformance.csv (Conceptual: StudentID, Gender, MathScore, ReadingScore, WritingScore, LunchType, TestPrepCourse). Assume some missing values and potential inconsistencies. (Instructor provides CSV).

Data Mining Tasks
Data Mining involves using automated or semi-automated techniques to explore and analyze large datasets to uncover meaningful patterns and rules. The specific goals can vary, leading to different types of tasks:

1. Classification
Goal: To assign items in a dataset to predefined target categories or classes. The model is trained on data where the classes are already known (supervised learning).
Example:

- Predicting whether an email is 'Spam' or 'Not Spam'.
- Determining if a customer is likely to 'Churn' or 'Not Churn'.
- Classifying a tumor as 'Benign' or 'Malignant' based on medical data.
- Predict category (e.g., Pass/Fail based on scores).

Output: A model that predicts a categorical class label for new, unseen data.

2. Regression
Goal: To predict a continuous numerical value, rather than a discrete class label. Like classification, it's a supervised learning task.
Example:

- Predicting the price of a house based on features like size, location, and age.
- Estimating the temperature tomorrow based on historical weather data.
- Predicting a student's test score based on study hours and previous grades.
- Predict continuous value (e.g., predict MathScore from ReadingScore).

Output: A model that predicts a continuous quantity for new data.

3. Clustering
Goal: To group similar items together into clusters based on their characteristics, without

prior knowledge of the groups (unsupervised learning). The goal is to maximize similarity within a cluster and minimize similarity between different clusters.
Example:
- Segmenting customers into different groups based on purchasing behavior for targeted marketing. • Grouping similar documents or news articles together.
- Identifying distinct communities within a social network.
- Group similar students without prior labels.

Output: A set of clusters, with data points assigned to each cluster.

4. Association Rule Mining (Market Basket Analysis)
Goal: To discover relationships or associations between items in large datasets. It identifies rules that describe how often items appear together.
Example:
- Identifying which products are frequently purchased together in an online store ("Customers who bought this also bought...").
- Discovering relationships between symptoms and diseases in medical records.
- Find links (e.g., students good at reading are often good at writing).

Output: Association rules, often in the form "If {Antecedent}, then {Consequent}", along with metrics like support and confidence.

5. Anomaly Detection (Outlier Detection)
Goal: To identify data points or events that are significantly different from the majority of the data or that deviate from expected patterns.
Example:
- Detecting fraudulent credit card transactions.
- Identifying faulty sensors or equipment based on unusual readings.
- Finding network intrusion attempts based on abnormal traffic patterns.

Output: Identification of data points flagged as anomalies or outliers.

6. Summarization
Goal: To provide a compact, concise description or overview of a dataset or a subset of it. This often involves visualization and descriptive statistics.
Example:
- Generating a report with key statistics like mean, median, standard deviation for sales data. • Creating visualizations (histograms, bar charts) to show the distribution of customer demographics. • Providing a textual summary of a long document.

Output: Condensed representations, statistics, or visualizations of the data.

7. Sequence Analysis (Sequential Pattern Mining)

Goal: To discover patterns or trends that occur in sequence over time. Similar to association rules but considers the order of events.

Example:

• Identifying the sequence of web pages a user typically visits before making a purchase. • Analyzing the order in which customers buy related products over multiple transactions. • Predicting the next likely purchase based on a customer's recent purchase history.

Output: Sequential patterns or rules indicating ordered relationships between events.

Data Preprocessing Necessity

Real-world data is often messy ("Garbage In, Garbage Out"). Preprocessing improves data quality for better mining results. Steps: Cleaning, Integration, Transformation, Reduction.

Data Cleaning: Filling missing values (imputation), smoothing noisy data, identifying and removing outliers, correcting inconsistencies (standardizing formats, resolving contradictions).

Data Integration: Combining data from multiple sources (databases, files) into a coherent dataset. This involves resolving schema differences, handling entity identification problems (ensuring 'Customer A' from source 1 is the same as 'Cust_A' from source 2), and managing data redundancy.

Data Transformation: Converting data into forms appropriate for mining. This includes:

Normalization/Standardization: Scaling numeric data to a common range (e.g., 0-1 or mean=0, stddev=1).

Attribute Construction: Creating new features from existing ones (e.g., calculating 'Age' from 'DateOfBirth').

Discretization: Converting continuous numeric data into intervals or categories. **Aggregation**: Summarizing data (e.g., calculating daily totals from hourly data).

Data Reduction: Obtaining a reduced representation of the dataset volume that produces similar analytical results. This is important for handling very large datasets. Techniques include: **Dimensionality Reduction:** Reducing the number of attributes/features (e.g., using Principal Component Analysis - PCA or removing irrelevant features).

Numerosity Reduction: Replacing the data with smaller representations (e.g., sampling, clustering, creating histograms).

Python/Pandas for Initial Exploration

1. Import Pandas: import pandas as pd
2. Load Data: Use pd.read_csv('your_file.csv').

```
import pandas as pd
```

```
# Make sure the CSV file is in the same directory as the notebook, or
provide the full path file_path = 'StudentPerformance.csv'
try:
 df = pd.read_csv(file_path)
 print("Dataset loaded successfully.")
except FileNotFoundError:
 print(f"Error: File not found at {file_path}")
 # Exit or handle error appropriately
```

3. Basic Inspection:
    a. .head(n): View the first n rows (default 5).

	StudentID	Gender	MathScore	ReadingScore	WritingScore	LunchType
0	1001	m	59.0	73.0	69.0	standard
1	1002	female	47.0	64.0	59.0	free/reduced
2	1003	male	85.0	83.0	86.0	standard
3	1004	male	76.0	66.0	64.0	standard
4	1005	female	47.0	59.0	60.0	free/reduced

    b. .tail(n): View the last n rows.
    c. .shape: Get dimensions (rows, columns).
    d. .info(): Get column data types, non-null counts, memory usage. Crucial for spotting type issues or preliminary missing values. [Image: Screenshot of df.info() output]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   StudentID       1000 non-null   int64
 1   Gender          969 non-null    object
 2   MathScore       951 non-null    float64
 3   ReadingScore    948 non-null    float64
 4   WritingScore    956 non-null    float64
 5   LunchType       955 non-null    object
 6   TestPrepCourse  927 non-null    object
dtypes: float64(3), int64(1), object(3)
memory usage: 54.8+ KB
None
```

    e. .describe(): Get descriptive statistics for numerical columns (count, mean, std dev, min, max, quartiles). Helps identify potential outliers or strange distributions.

```
         StudentID   MathScore  ReadingScore  WritingScore
count  1000.000000  951.000000    948.000000    956.000000
mean   1500.500000   59.041009     60.955696     60.781381
std     288.819436   18.235979     17.918039     18.333139
min    1001.000000   21.000000     22.000000     21.000000
25%    1250.750000   44.000000     46.000000     46.750000
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
```

f. .describe(include='object'): Get stats for categorical columns (count, unique values, top value, frequency).

```
        Gender LunchType TestPrepCourse
count      969       955            927
unique       6         5              4
top     female  standard           none
freq       459       562            646
```

g. .columns: List column names.

h. .dtypes: Check data types of each column.

```
StudentID           int64
Gender             object
MathScore         float64
ReadingScore      float64
WritingScore      float64
LunchType          object
TestPrepCourse     object
dtype: object
```

4. Checking for Missing Values:
   a. .isnull(): Returns a DataFrame of booleans (True if value is missing/NaN).
   b. .isnull().sum(): Counts missing values per column. Very useful!


```
StudentID          0
Gender            31
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
```

```
# Check for missing values
missing_values = df.isnull().sum()
print("\nMissing values per column:")
print(missing_values[missing_values > 0]) # Only show columns
```

with missing values 5. Checking Unique Values: For categorical

columns, see the distinct values present.

```
# Check unique values in a categorical column
print("\nUnique values in 'Gender':")
print(df['Gender'].unique())
print("\nValue counts for 'LunchType':")
print(df['LunchType'].value_counts())
```

Exercises:

1. Import the Pandas library.
2. Load the StudentPerformance.csv dataset into a Pandas DataFrame called student_df. Handle potential FileNotFoundError.
3. Display the first 7 rows and the last 7 rows of the DataFrame.
4. Display the dimensions (number of rows and columns) of the DataFrame.
5. Display the summary information (using .info()) - pay attention to data types and non-null counts. 6.Calculate and display the descriptive statistics for numerical columns. Does anything look unusual (e.g., scores outside 0-100)?
7. Calculate and display the number of missing values for each column in the DataFrame. 8.List the unique values found in the TestPrepCourse column.
9. Based only on the output of .info() and .isnull().sum(), list the names of the columns that definitely require some preprocessing before being used in most standard data mining algorithms. Explain why for each column listed.

Code Snippet

```
import pandas as pd
try:
   student_df = pd.read_csv("StudentPerformance.csv")
except FileNotFoundError:
   print(" File not found. Make sure 'StudentPerformance.csv' is in the same folder.")
   exit()
print(" First 7 rows:\n", student_df.head(7))
```

```
print(" Last 7 rows:\n", student_df.tail(7))
print(" DataFrame shape:", student_df.shape)
print(" Summary info:")
student_df.info()
print(" Descriptive statistics:")
print(student_df.describe())
print(" Missing values per column:\n", student_df.isnull().sum())
if "TestPrepCourse" in student_df.columns:
    print(" Unique TestPrepCourse values:", student_df["TestPrepCourse"].unique())
else:
    print(" Column 'TestPrepCourse' not found.")
print("\n Based on data info and missing values:")
print("- Handle missing values in columns that are not 100% non-null.")
print("- Convert categorical columns like gender, race/ethnicity, parental level of education into numeric using
   encoding.")
```

Outputs

```
- Convert categorical columns like gender, race/ethnicity, parental level of education into numeric using en
PS C:\Users\user\Downloads\labs dwm> python lab8.py
 First 7 rows:
   StudentID  Gender  MathScore  ReadingScore  WritingScore     LunchType TestPrepCourse
0       1001       m       59.0          73.0          69.0      standard      completed
1       1002  female       47.0          64.0          59.0  free/reduced           none
2       1003    male       85.0          83.0          86.0      standard           none
3       1004    male       76.0          66.0          64.0      standard            NaN
4       1005  female       47.0          59.0          60.0  free/reduced      completed
5       1006    male       77.0          80.0          83.0           NaN           none
6       1007  female       39.0          38.0          39.0  free/reduced      completed
 Last 7 rows:
     StudentID  Gender  MathScore  ReadingScore  WritingScore     LunchType TestPrepCourse
993       1994    male       64.0          70.0          72.0      standard           none
994       1995  female       39.0          52.0          50.0      standard      completed
995       1996  female       78.0           NaN          71.0      standard      completed
996       1997    male       51.0          59.0          59.0  free/reduced      Completed
997       1998    male       74.0          72.0          71.0      standard           none
998       1999  female       65.0          70.0          68.0      standard           none
999       2000    Male       70.0          74.0          69.0      standard           none
 DataFrame shape: (1000, 7)
 Summary info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   StudentID       1000 non-null   int64
 1   Gender          969 non-null    object
 2   MathScore       951 non-null    float64
 3   ReadingScore    948 non-null    float64
 4   WritingScore    956 non-null    float64
 5   LunchType       955 non-null    object
 6   TestPrepCourse  927 non-null    object
dtypes: float64(3), int64(1), object(3)
memory usage: 54.8+ KB
 Descriptive statistics:
         StudentID   MathScore  ReadingScore  WritingScore
count  1000.000000  951.000000    948.000000    956.000000
mean   1500.500000   59.041009     60.955696     60.781381
std     288.819436   18.235979     17.918039     18.333139
min    1001.000000   21.000000     22.000000     21.000000
25%    1250.750000   44.000000     46.000000     46.750000
```

```
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']

 Based on data info and missing values:
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
 Missing values per column:
 StudentID         0
Gender            31
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
 Missing values per column:
 StudentID         0
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
 Missing values per column:
 StudentID         0
Gender            31
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
50%    1500.500000   59.000000     62.000000     61.000000
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
 Missing values per column:
 StudentID         0
Gender            31
```

```
- Handle missing values in columns that are not 100% non-null.
75%    1750.250000   75.000000     76.000000     76.000000
max    2000.000000   99.000000    100.000000    100.000000
 Missing values per column:
 StudentID         0
Gender            31
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
 Missing values per column:
 StudentID         0
Gender            31
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
MathScore         49
ReadingScore      52
WritingScore      44
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
LunchType         45
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
TestPrepCourse    73
dtype: int64
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']
 Unique TestPrepCourse values: ['completed' 'none' nan 'Completed' 'not completed']

 Based on data info and missing values:
- Handle missing values in columns that are not 100% non-null.
- Convert categorical columns like gender, race/ethnicity, parental level of education into numeric using encoding.
PS C:\Users\user\Downloads\labs dwm>
```

Lab Session 09

Preprocessing Techniques using Python

Objective
Apply common preprocessing techniques in Python: missing value imputation and data transformation (scaling, encoding).

Dataset
StudentPerformance.csv loaded into student_df DataFrame from Lab 8.

Data Cleaning

Standardize Categories: Convert uppercase to lowercase, Removes leading/trailing whitespace, Maps the abbreviations etc. 'f' and 'm' to their full forms.
Imputation: Replacing missing values (NaN). Common strategies:

- **Mean**: df['column'].fillna(df['column'].mean(), inplace=True) (Good for symmetric data, sensitive to outliers).
- **Median**: df['column'].fillna(df['column'].median(), inplace=True) (Better for skewed data or with outliers).
- **Mode**: df['column'].fillna(df['column'].mode()[0], inplace=True) (Good for categorical data, can be used for numerical). [0] is needed as .mode() can return multiple values if tied.

```
from sklearn.impute import SimpleImputer
import numpy as np

# Impute numerical columns with median
num_cols = ['MathScore', 'ReadingScore', 'WritingScore']
imputer_median = SimpleImputer(missing_values=np.nan, strategy='median')
# Use .fit_transform on the subset of columns, assign back
df[num_cols] = imputer_median.fit_transform(df[num_cols])

# Standardize: Lowercase, trim whitespace and map abbreviations
col_name = 'Gender'
df[col_name] = df[col_name].str.lower()
df[col_name] = df[col_name].str.strip()
```

```
gender_map = {'f': 'female', 'm': 'male'}
df[col_name] = df[col_name].replace(gender_map)

col_name = 'LunchType'
df[col_name] = df[col_name].str.lower()
df[col_name] = df[col_name].str.strip()

col_name = 'TestPrepCourse'
df[col_name] = df[col_name].str.lower()
df[col_name] = df[col_name].str.strip()

# Impute categorical columns with mode (if any missing)
 cat_cols = ['Gender', 'LunchType', 'TestPrepCourse'] # Define categorical columns
 imputer_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
 df[cat_cols] = imputer_mode.fit_transform(df[cat_cols])

# Verify imputation
print("Missing values after imputation:")
print(df.isnull().sum())
```

Data Transformation:

Scaling Numerical Data: Needed for distance-based algorithms (K-Means, KNN, SVM) or gradient-based optimization.

Min-Max Scaling (Normalization): Scales to [0, 1]. Use MinMaxScaler.

```
from sklearn.preprocessing import MinMaxScaler
scaler_minmax = MinMaxScaler()
# Apply only to numerical columns intended for scaling
df[num_cols] = scaler_minmax.fit_transform(df[num_cols])
print("\nFirst 5 rows after Min-Max Scaling:")
print(df[num_cols].head())
```

Standardization (Z-score): Scales to mean=0, stddev=1. Use StandardScaler. Less affected by outliers than Min Max.

```
from sklearn.preprocessing import StandardScaler
scaler_standard = StandardScaler()
# df[num_cols] = scaler_standard.fit_transform(df[num_cols]) # Apply if needed
```

Encoding Categorical Data:
Converting categories to numbers.
One-Hot Encoding: Creates new binary (0/1) columns for each category. Prevents implying order. Use pd.get_dummies()

```
# Example for 'Gender' , 'LunchType', and , 'TestPrepCourse' using pandas
df_encoded = pd.get_dummies(df, columns=['Gender', 'LunchType',
'TestPrepCourse'], drop_first=True)  # drop_first=True avoids multicollinearity by
dropping one category per feature

print("\nDataFrame columns after One-Hot Encoding:")
print(df_encoded.columns)
print("\nFirst 5 rows of encoded DataFrame:")
print(df_encoded.head())
```

Label Encoding: Assigns a unique integer to each category (e.g., Male=0, Female=1). Implies order, suitable for ordinal features or tree-based models. Use LabelEncoder.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Example for 'TestPrepCourse'
df['TestPrepCourse_encoded'] = le.fit_transform(df['TestPrepCourse'])
print("\nFirst 5 rows of encoded DataFrame:")
print(df[['TestPrepCourse', 'TestPrepCourse_encoded']].head())
```

Exercises

1. Load the StudentPerformance.csv data into a DataFrame df. Make a copy df_processed = df.copy() to work on.

2. Defined the numerical score columns (MathScore, ReadingScore, WritingScore). 3. Impute any missing values in these numerical columns using the median strategy. Use either Pandas .fillna() or Scikit-learn's SimpleImputer. Verify that there are no more missing values in these columns.

4. Apply Standardization (Z-score scaling) to the imputed numerical score columns. Store the results back into the df_processed DataFrame. Display the first 5 rows of the processed scores. 5. Defined the categorical columns (Gender, LunchType, TestPrepCourse).

6. Apply One-Hot Encoding to these categorical columns using pd.get_dummies(). Make sure to handle potential multicollinearity (drop_first=True). Store the result in a new DataFrame df_encoded. Display the column names and the first 5 rows of df_encoded. How many columns does it have compared to df_processed?

Code Snipet

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("StudentPerformance.csv")
df_processed = df.copy()
num_cols = ['MathScore', 'ReadingScore', 'WritingScore']
imputer = SimpleImputer(strategy='median')
df_processed[num_cols] = imputer.fit_transform(df_processed[num_cols])
print("Missing values in numerical columns after imputation:\n", df_processed[num_cols].isnull().sum())
scaler = StandardScaler()
df_processed[num_cols] = scaler.fit_transform(df_processed[num_cols])
print("\nStandardized numerical scores (first 5 rows):\n", df_processed[num_cols].head())
cat_cols = ['Gender', 'LunchType', 'TestPrepCourse']
df_encoded = pd.get_dummies(df_processed[cat_cols], drop_first=True)
print("\nEncoded categorical columns:\n", df_encoded.columns)
print(df_encoded.head())
print(f"\nOriginal processed DataFrame columns: {df_processed.shape[1]}")
print(f"Encoded DataFrame columns: {df_encoded.shape[1]}")
```

```
PS C:\Users\user\Downloads\labs dwm> python lab9.py
Missing values in numerical columns after imputation:
 MathScore       0
ReadingScore    0
WritingScore    0
dtype: int64

Standardized numerical scores (first 5 rows):
    MathScore  ReadingScore  WritingScore
0  -0.002194      0.687567      0.458195
1  -0.677329      0.171462     -0.099967
2   1.460597      1.261018      1.407070
3   0.954246      0.286152      0.179114
4  -0.677329     -0.115264     -0.044151

Encoded categorical columns:
 Index(['Gender_Male', 'Gender_f', 'Gender_female', 'Gender_m', 'Gender_male',
       'LunchType_Standard', 'LunchType_free/reduced', 'LunchType_standard',
       'LunchType_standard ', 'TestPrepCourse_completed',
       'TestPrepCourse_none', 'TestPrepCourse_not completed'],
      dtype='object')
   Gender_Male  Gender_f  Gender_female  Gender_m  ...  LunchType_standard   TestPrepCourse_completed  TestPrepCourse_none  TestPrepCourse_not completed
0        False     False          False      True  ...                False                       True                False                         False
1        False     False           True     False  ...                False                      False                 True                         False
2        False     False          False     False  ...                False                      False                 True                         False
3        False     False          False     False  ...                False                      False                False                         False
4        False     False           True     False  ...                False                       True                False                         False

[5 rows x 12 columns]

Original processed DataFrame columns: 7
Encoded DataFrame columns: 12
PS C:\Users\user\Downloads\labs dwm>
```

Lab Session 10

Classification I: Decision Trees

Objective
Apply the Decision Tree algorithm for binary classification. Understand how to build, visualize, and evaluate a decision tree model in Python.

Dataset
Preprocessed StudentPerformance data from Lab 9 (imputed scores, potentially encoded categorical features).

- **Target Variable**: We will create a binary variable PassedMath ('1' if MathScore >= 50, '0' otherwise).
- **Features**: Use preprocessed ReadingScore, WritingScore, and the one-hot encoded features for Gender, LunchType, TestPrepCourse.

Classification Overview
Predicting a categorical target variable. Decision Trees are a non-parametric supervised learning method used for both classification and regression. They work by learning simple decision rules inferred from the data features.

Decision Tree Algorithm Basics
Predicting a categorical target variable. Decision Trees are a non-parametric supervised learning method used for both classification and regression

- Partitions the data into subsets based on feature values.
- Nodes: Root node (entire population), Decision nodes (split based on a feature), Leaf nodes (final outcome/class label).
- Splitting Criteria: Algorithms like ID3 (uses Entropy and Information Gain) or CART (uses Gini Impurity) decide the best feature and threshold to split on at each node.

Steps in Python (Scikit-learn):
Predicting a categorical target variable. Decision Trees are a non-parametric supervised learning method used for both classification and regression
Prepare Data:

- Load your preprocessed DataFrame (e.g., df_encoded from Lab 9).
- Create the binary target variable.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Optional for visualization
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

```
# Make sure the CSV file is in the same directory as the notebook, or
provide the full path file_path = 'StudentPerformance_cleaned.csv'
try:
 df = pd.read_csv(file_path)
 df['PassedMath'] = (df['MathScore'] >= 50).astype(int)


 print("DataFrame with target variable 'PassedMath':")
 print(df.head())

except FileNotFoundError:
 print(f"Error: File not found at {file_path}")
 # Exit or handle error appropriately
```

• Define features (X) and target (y).

```
 # Features: Exclude MathScore (if target is derived from it)
  feature_columns = ['ReadingScore', 'WritingScore', 'Gender_male',
 'LunchType_standard', 'TestPrepCourse_none', 'TestPrepCourse_not
completed']  X = df[feature_columns]
 y = df['PassedMath']
 print("\nFeatures (X) head:")
 print(X.head())
 print("\nTarget (y) head:")
 print(y.head())
```

Split Data:
Divide into training and testing sets.

```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)  # stratify=y is good for classification to
maintain class proportion in splits
 print(f"\nShape of X_train: {X_train.shape}, Shape of y_train: {y_train.shape}")
 print(f"Shape of X_test: {X_test.shape}, Shape of y_test: {y_test.shape}")
```

Train Model:
Instantiate DecisionTreeClassifier and fit it to the training data.

```
# Initialize the Decision Tree Classifier
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
# Train the model
dt_model.fit(X_train, y_train)
print("\nDecision Tree model trained.")
```

Make Predictions:

```
y_pred = dt_model.predict(X_test)
    print("\nPredictions made on the test set.")
```

Evaluate Model:

Accuracy: Proportion of correct predictions.
(TP+TN)/(TP+TN+FP+FN)

Confusion Matrix: Shows TP (True Positives), TN (True Negatives), FP (False Positives), FN (False Negatives).
[[TN, FP], [FN, TP]]

Classification Report: Precision, Recall, F1-score per class.
Precision = TP / (TP + FP) (accuracy of positive predictions)
Recall (Sensitivity) = TP / (TP + FN) (ability to find all positive samples)

F1-score = 2 * (Precision * Recall) / (Precision + Recall) (harmonic mean)

```
print(f"\nAccuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
# For better display of confusion matrix:
import seaborn as sns
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Fail', 'Pass'],  yticklabels=['Fail', 'Pass'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
print("\nClassification Report:")
print(class_report)
```

Exercise

1. Load your preprocessed student dataset (df_encoded or similar from Lab 9).
2. Create the binary target variable PassedReading (1 if original ReadingScore >= 50, 0 otherwise). Ensure you use the original or unscaled reading score for this, not the scaled one.
3. Define your feature matrix X (using preprocessed MathScore_scaled, WritingScore_scaled, and one-hot encoded Gender, LunchType, TestPrepCourse) and target vector y (PassedReading).
4. Split your data into training (70%) and testing (30%) sets. Use random_state=42 and stratify=y. 5. Train a DecisionTreeClassifier model on the training data. Experiment with max_depth values (e.g., 3, 5, None). For the final model, use max_depth=4 and random_state=42.
6. Make predictions on the test data.
7. Evaluate the model:
   - Calculate and print the accuracy.
   - Generate and print the confusion matrix.
   - Generate and print the classification report.

PYTHON SCRIPT:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler

# Loading and preprocessing the dataset
file_path = 'StudentPerformance.csv'
df = pd.read_csv(file_path)

# Impute numerical columns with median
num_cols = ['MathScore', 'WritingScore']
imputer_median = SimpleImputer(missing_values=np.nan, strategy='median')
df[num_cols] = imputer_median.fit_transform(df[num_cols])

# Standardize categorical columns
col_name = 'Gender'
df[col_name] = df[col_name].str.lower().str.strip()
gender_map = {'f': 'female', 'm': 'male'}
df[col_name] = df[col_name].replace(gender_map)
```

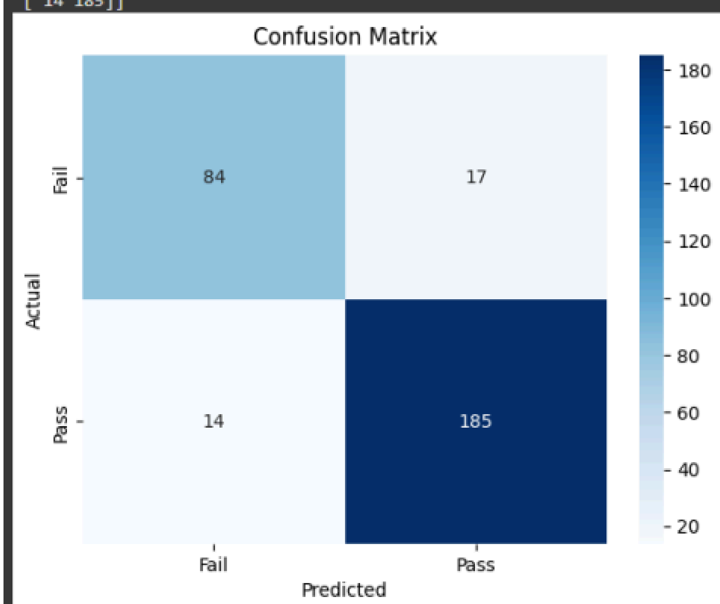
```
col_name = 'LunchType'
df[col_name] = df[col_name].str.lower().str.strip()

col_name = 'TestPrepCourse'
df[col_name] = df[col_name].str.lower().str.strip()

# Impute categorical columns with mode
cat_cols = ['Gender', 'LunchType', 'TestPrepCourse']
imputer_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df[cat_cols] = imputer_mode.fit_transform(df[cat_cols])

# Creating binary target variable
df['PassedReading'] = (df['ReadingScore'] >= 50).astype(int)

# Scaling numerical features using MinMaxScaler
scaler = MinMaxScaler()
df[['MathScore_scaled', 'WritingScore_scaled']] = scaler.fit_transform(df[['MathScore',
'WritingScore']])

# One-hot encoding categorical variables
df_encoded = pd.get_dummies(df, columns=['Gender', 'LunchType', 'TestPrepCourse'],
drop_first=True)

# Defining feature matrix X and target vector y
X = df_encoded[['MathScore_scaled', 'WritingScore_scaled',
           'Gender_male', 'LunchType_standard',
           'TestPrepCourse_none', 'TestPrepCourse_not completed']]
print("\nFeatures (X) head:")
print(X.head())
y = df_encoded['PassedReading']
print("\nTarget (y) head:")
print(y.head())

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=42,
                                                    stratify=y)
print(f"\nShape of X_train: {X_train.shape}, Shape of y_train: {y_train.shape}")
print(f"Shape of X_test: {X_test.shape}, Shape of y_test: {y_test.shape}")

# Training DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(max_depth=4, random_state=42)
dt_classifier.fit(X_train, y_train)

# Making predictions
```

```
y_pred = dt_classifier.predict(X_test)

# Evaluating the model
dt_classifier.fit(X_train, y_train)
print("\nDecision Tree model trained.")
y_pred = dt_classifier.predict(X_test)
print("\nPredictions made on the test set.")
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Printing results
print("Model Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)

# [Image: Seaborn Confusion Matrix plot]
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Fail', 'Pass'],
yticklabels=['Fail', 'Pass'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()


print("\nClassification Report:")
print(class_report)

plt.figure(figsize=(20,10)) # Adjust figure size
plot_tree(dt_classifier,
        feature_names=X.columns.tolist(),
        class_names=['Fail (0)', 'Pass (1)'], #Ensure order matches class labels
        filled=True,
        rounded=True,
        fontsize=10)
plt.title("Decision Tree for Reading Performance")
plt.show()
```

OUTPUT:

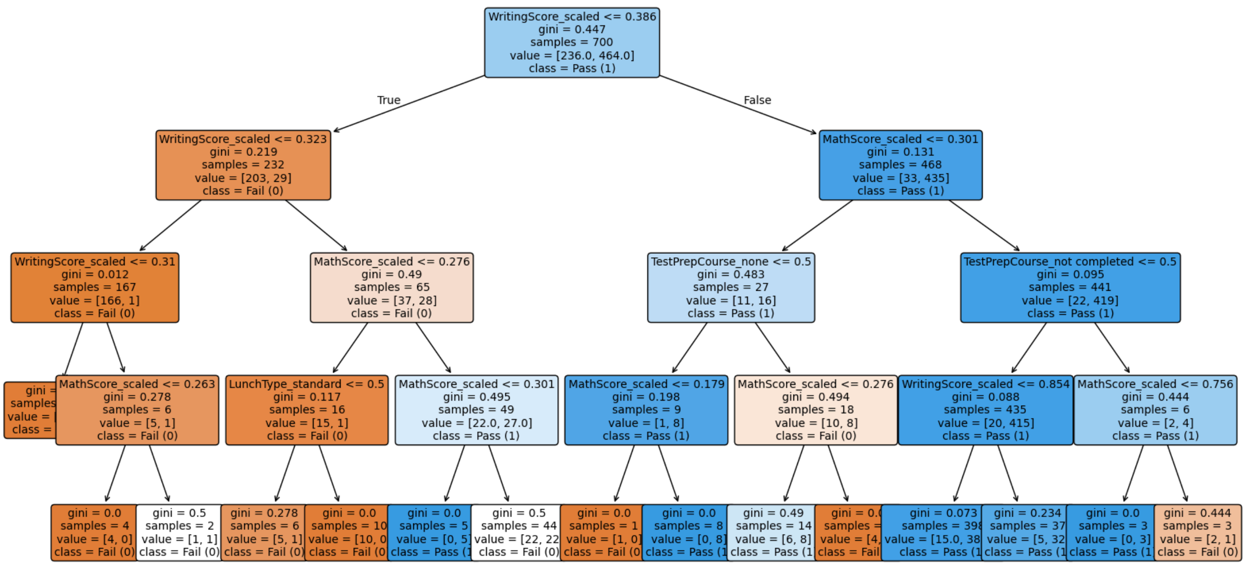
```
Features (X) head:
   MathScore_scaled  WritingScore_scaled  Gender_male  LunchType_standard  \
0          0.487179             0.607595         True                True
1          0.333333             0.481013        False               False
2          0.820513             0.822785         True                True
3          0.705128             0.544304         True                True
4          0.333333             0.493671        False               False

   TestPrepCourse_none  TestPrepCourse_not completed
0                False                         False
1                 True                         False
2                 True                         False
3                 True                         False
4                False                         False

Target (y) head:
0    1
1    1
2    1
3    1
4    1
Name: PassedReading, dtype: int64

Shape of X_train: (700, 6), Shape of y_train: (700,)
Shape of X_test: (300, 6), Shape of y_test: (300,)

Decision Tree model trained.
```

```
Predictions made on the test set.
Model Accuracy: 0.8966666666666666

Confusion Matrix:
[[ 84  17]
 [ 14 185]]
```



```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.83      0.84       101
           1       0.92      0.93      0.92       199

    accuracy                           0.90       300
   macro avg       0.89      0.88      0.88       300
weighted avg       0.90      0.90      0.90       300
```

Decision Tree for Reading Performance



Lab Session 11

Classification II: Naive Bayes with Python

Objective

Apply the Naive Bayes algorithm for binary classification and compare its performance with the Decision Tree model.

Dataset

Same preprocessed StudentPerformance data and target variable (PassedMath or PassedReading from Lab 10) and feature set.

Naive Bayes Algorithm Basics

Probabilistic classifier based on Bayes' Theorem.
"Naive" Assumption: Assumes that all features are independent of each other given the class. This simplifies computation but might not hold true in real-world data.

Types:

- **GaussianNB**: Assumes continuous features follow a Gaussian (normal) distribution. Suitable for features like scaled scores.
- **MultinomialNB**: Suitable for discrete features (e.g., word counts in text classification).
- **BernoulliNB**: Suitable for binary/boolean features (e.g., one-hot encoded features, if treated as presence/absence).

Steps in Python:

1. Prepare Data: Ensure features (X) and target (y) are ready, and training/testing sets (X_train, X_test, y_train, y_test) are already created (can reuse from Lab 10).

```
# Assuming X_train, X_test, y_train, y_test are available from Lab 10
# (using the 'PassedMath' prediction task as an example)
# X_train, X_test, y_train, y_test from Lab 10's PassedMath section
print("Using existing train/test splits from Lab 10 (PassedMath example).")
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
```

2. Train Model: Instantiate GaussianNB and fit it.

```
from sklearn.naive_bayes import GaussianNB

# Initialize Gaussian Naive Bayes classifier
nb_model = GaussianNB()

# Train the model
```

```
nb_model.fit(X_train, y_train)
print("\nGaussian Naive Bayes model trained.")
```

3. Make Predictions:

```
y_pred_nb = nb_model.predict(X_test)
print("\nPredictions made on the test set using Naive Bayes.")
```

4. Evaluate Model: Same metrics as for Decision Trees.

```
accuracy_nb = accuracy_score(y_test, y_pred_nb)
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)
class_report_nb = classification_report(y_test, y_pred_nb)

print(f"\nNaive Bayes Accuracy: {accuracy_nb:.4f}")
print("\nNaive Bayes Confusion Matrix:")
print(conf_matrix_nb)
# Optionally display with Seaborn as in Lab 10
# [Image: Seaborn Confusion Matrix plot for Naive Bayes]
print("\nNaive Bayes Classification Report:")
print(class_report_nb)
```

5. Comparison: Compare metrics (accuracy, precision, recall, F1) against the Decision Tree model from Lab 10 for the same target variable and data split.

Exercise:

1. Using the same training (X_train_r, y_train_r) and testing sets (X_test_r, y_test_r) that you created for the PassedReading target variable in Lab 10, Exercise 4.

Code

lab11.py > ...

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'StudentPerformance_cleaned.csv'

try:
    df = pd.read_csv(file_path)

    # Create binary target: PassedReading (1 = MathScore >= 50)
    df['PassedReading'] = (df['MathScore'] >= 50).astype(int)
    print("Target variable 'PassedReading' added.\n")
    print(df.head())

    # Select features
    feature_columns = ['ReadingScore', 'WritingScore', 'Gender_male',
                       'LunchType_standard', 'TestPrepCourse_none', 'TestPrepCourse_not completed']

    X = df[feature_columns]
    y = df['PassedReading']

    print("\nFeatures (X) head:")
    print(X.head())
    print("\nTarget (y) head:")
    print(y.head())

    # Split data into training and test sets
    X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y)

    print(f"\nTraining and testing data prepared. Train shape: {X_train_r.shape}")

    # Train Gaussian Naive Bayes model
    nb_model = GaussianNB()
    nb_model.fit(X_train_r, y_train_r)
    print("\nGaussian Naive Bayes model trained.")

```

```python
    print(f"\nTraining and testing data prepared. Train shape: {X_train_r.shape}")

    # Train Gaussian Naive Bayes model
    nb_model = GaussianNB()
    nb_model.fit(X_train_r, y_train_r)
    print("\nGaussian Naive Bayes model trained.")

    # Make predictions with Naive Bayes
    y_pred_nb = nb_model.predict(X_test_r)
    print("Predictions made with Naive Bayes.")

    # Evaluate Naive Bayes model
    accuracy_nb = accuracy_score(y_test_r, y_pred_nb)
    conf_matrix_nb = confusion_matrix(y_test_r, y_pred_nb)
    class_report_nb = classification_report(y_test_r, y_pred_nb)

    print(f"\nNaive Bayes Accuracy: {accuracy_nb:.4f}")
    print("\nNaive Bayes Confusion Matrix:")
    print(conf_matrix_nb)
    print("\nNaive Bayes Classification Report:")
    print(class_report_nb)

    # Confusion matrix heatmap
    sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap='Purples',
                xticklabels=['Fail', 'Pass'], yticklabels=['Fail', 'Pass'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Naive Bayes Confusion Matrix (PassedReading)')
    plt.show()

    # Train Decision Tree model (max_depth=4)
    dt_model = DecisionTreeClassifier(max_depth=4, random_state=42)
    dt_model.fit(X_train_r, y_train_r)
    y_pred_dt = dt_model.predict(X_test_r)

    accuracy_dt = accuracy_score(y_test_r, y_pred_dt)
    class_report_dt = classification_report(y_test_r, y_pred_dt)

    print(f"\nDecision Tree Accuracy (max_depth=4): {accuracy_dt:.4f}")
    print("\nDecision Tree Classification Report:")
    print(class_report_dt)
```

```python
    # Compare Precision & Recall for class '1' (Passed)
    print("\nComparison for Class '1' (Passed):")

    precision_nb = precision_score(y_test_r, y_pred_nb)
    recall_nb = recall_score(y_test_r, y_pred_nb)
    precision_dt = precision_score(y_test_r, y_pred_dt)
    recall_dt = recall_score(y_test_r, y_pred_dt)

    print(f"Naive Bayes - Precision: {precision_nb:.4f}, Recall: {recall_nb:.4f}")
    print(f"Decision Tree - Precision: {precision_dt:.4f}, Recall: {recall_dt:.4f}")

    print("\nComment:")
    if precision_nb > precision_dt:
        print("- Naive Bayes gives better precision for 'Passed'.")
    else:
        print("- Decision Tree gives better precision for 'Passed'.")

    if recall_nb > recall_dt:
        print("- Naive Bayes gives better recall for 'Passed'.")
    else:
        print("- Decision Tree gives better recall for 'Passed'.")

except FileNotFoundError:
    print(f"Error: File not found at {file_path}")
```

Output

```
   StudentID  MathScore  ReadingScore  ...  TestPrepCourse_none  TestPrepCourse_not completed  PassedReading
0       1001       59.0      0.653846  ...                False                        False              1
1       1002       47.0      0.538462  ...                 True                        False              0
2       1003       85.0      0.782051  ...                 True                        False              1
3       1004       76.0      0.564103  ...                 True                        False              1
4       1005       47.0      0.474359  ...                False                        False              0

[5 rows x 9 columns]

Features (X) head:
   ReadingScore  WritingScore  Gender_male  LunchType_standard  TestPrepCourse_none  TestPrepCourse_not completed
0      0.653846      0.607595         True                True                False                         False
1      0.538462      0.481013        False               False                 True                         False
2      0.782051      0.822785         True                True                 True                         False
3      0.564103      0.544304         True                True                 True                         False
4      0.474359      0.493671        False               False                False                         False

Target (y) head:
0    1
1    0
2    1
3    1
4    0
Name: PassedReading, dtype: int64

Training and testing data prepared. Train shape: (700, 6)

Gaussian Naive Bayes model trained.
Predictions made with Naive Bayes.
```

2. Train a GaussianNB classifier model on this training data.

Code
Step 1: Train Gaussian Naive Bayes model nb_model = GaussianNB() nb_model.fit(X_train_r, y_train_r) print("\nGaussian Naive Bayes model trained.")

Output

```
Gaussian Naive Bayes model trained.
```

3. Make predictions on the corresponding test data.

Code
Step 2: Make predictions y_pred_nb = nb_model.predict(X_test_r) print("Predictions made with Naive Bayes.")

Output

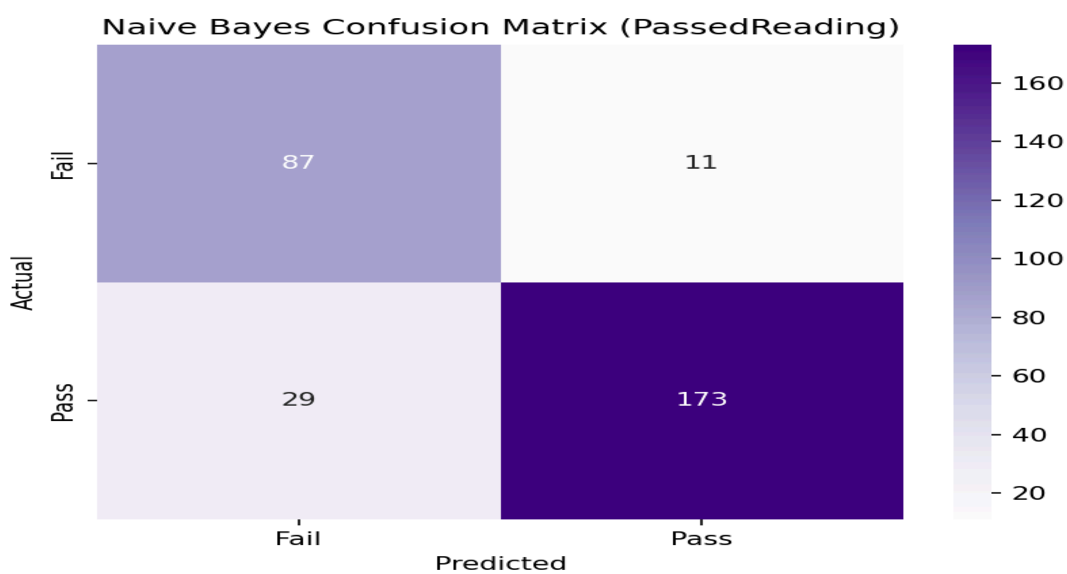
```
Predictions made with Naive Bayes.
```

4. Evaluate the Naive Bayes model:
   Calculate and print the accuracy.
   Generate and print the confusion matrix.
   Generate and print the classification report.

Code
Step 3: Evaluate Naive Bayes accuracy_nb = accuracy_score(y_test_r, y_pred_nb) conf_matrix_nb = confusion_matrix(y_test_r, y_pred_nb) class_report_nb = classification_report(y_test_r, y_pred_nb) print(f"\nNaive Bayes Accuracy: {accuracy_nb:.4f}") print("\nNaive Bayes Confusion Matrix:") print(conf_matrix_nb) print("\nNaive Bayes Classification Report:") print(class_report_nb)

Output

```
Naive Bayes Confusion Matrix:
[[ 87  11]
 [ 29 173]]

Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.89      0.81        98
           1       0.94      0.86      0.90       202

    accuracy                           0.87       300
   macro avg       0.85      0.87      0.85       300
weighted avg       0.88      0.87      0.87       300


Decision Tree Accuracy (max_depth=4): 0.8933
```



5. Compare the overall accuracy of the GaussianNB model for PassedReading with the accuracy of your DecisionTreeClassifier (with max_depth=4) for PassedReading from Lab 10.

Code

Step 4: Compare with Decision Tree (max_depth=4) dt_model = DecisionTreeClassifier(max_depth=4, random_state=42) dt_model.fit(X_train_r, y_train_r) y_pred_dt = dt_model.predict(X_test_r) accuracy_dt = accuracy_score(y_test_r, y_pred_dt) class_report_dt = classification_report(y_test_r, y_pred_dt) print(f"\nDecision Tree Accuracy (max_depth=4): {accuracy_dt:.4f}") print("\nDecision Tree Classification Report:") print(class_report_dt)

Output

```
Decision Tree Accuracy (max_depth=4): 0.8933

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.83      0.84        98
           1       0.92      0.93      0.92       202

    accuracy                           0.89       300
   macro avg       0.88      0.88      0.88       300
weighted avg       0.89      0.89      0.89       300
```

6. Look at the precision and recall for Class '1' (Passed) for both models. Which model gives better precision for 'Passed'? Which gives better recall for 'Passed'? Briefly comment on what this means.

Code

Step 5: Precision & Recall comparison for class '1' (Passed) print("\nComparison for Class '1' (Passed):") from sklearn.metrics import precision_score, recall_score precision_nb = precision_score(y_test_r, y_pred_nb) recall_nb = recall_score(y_test_r, y_pred_nb) precision_dt = precision_score(y_test_r, y_pred_dt) recall_dt = recall_score(y_test_r, y_pred_dt) print(f"Naive Bayes - Precision: {precision_nb:.4f}, Recall: {recall_nb:.4f}") print(f"Decision Tree - Precision: {precision_dt:.4f}, Recall: {recall_dt:.4f}")

Output

```
Comparison for Class '1' (Passed):
Naive Bayes - Precision: 0.9402, Recall: 0.8564
Decision Tree - Precision: 0.9167, Recall: 0.9257

Comment:
- Naive Bayes gives better precision for 'Passed'.
- Decision Tree gives better recall for 'Passed'.
```

Naive Bayes vs Decision Tree (Class 'Passed'):

- **Precision:**
  - Naive Bayes: **0.9402**
  - Decision Tree: 0.9167
    → *Naive Bayes is better at avoiding false positives.*
- **Recall:**
  - Naive Bayes: 0.8564
  - Decision Tree: **0.9257**
    → *Decision Tree is better at catching actual pass cases.*

Lab Session 12

Clustering: K-Means Algorithm

Objective

Apply the K-Means clustering algorithm to segment data. Understand how to determine K using the Elbow Method and visualize clusters.

Dataset

Customer_Segmentation.csv (Conceptual - Instructor provides CSV). Columns: CustomerID, Age, AnnualIncome (in thousands, e.g., 15-150), SpendingScore (1-100). We will cluster based on AnnualIncome and SpendingScore.

Clustering Overview

Unsupervised learning task to group similar data points without predefined labels. K-Means is a popular partitioning method.

K-Means Algorithm Basics:

1. Choose K (number of clusters).
2. Initialize K centroids randomly.
3. Assignment Step: Assign each data point to the nearest centroid.
4. Update Step: Recalculate centroids as the mean of points assigned to them.
5. Repeat steps 3-4 until convergence (centroids don't change much or max iterations reached).

Importance of Scaling:

K-Means uses Euclidean distance. Features with larger ranges can dominate. Standardize/Normalize features before K-Means.

Steps in Python (Scikit-learn):

Customer_Segmentation.csv (Conceptual - Instructor provides CSV). Columns: CustomerID, Age, AnnualIncome (in thousands, e.g., 15-150), SpendingScore (1-100). We will cluster based on AnnualIncome and SpendingScore.

Load and Prepare Data:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
# Assume 'Customer_Segmentation.csv' has CustomerID, Age,
AnnualIncome, SpendingScore try:
 customer_df = pd.read_csv('Customer_Segmentation.csv')
 print("Customer dataset loaded successfully.")
```

```
except FileNotFoundError:
 print("Error: Customer_Segmentation.csv not found.")
 # Create a dummy df for continuation of example if file not found by user
 customer_df = pd.DataFrame({
 'CustomerID': range(1, 201),
 'Age': np.random.randint(18, 70, 200),
 'AnnualIncome': np.random.randint(15, 150, 200),
 'SpendingScore': np.random.randint(1, 100, 200) })
 print("Created a dummy customer dataset.")

print("\nCustomer Data Head:")
print(customer_df.head())

# Select features for clustering
# For this lab, we focus on 'AnnualIncome' and 'SpendingScore'
X_customer = customer_df[['AnnualIncome', 'SpendingScore']]
print("\nSelected features for clustering (X_customer head):")
print(X_customer.head())
```

Scale Features:

```
scaler = StandardScaler()
X_customer_scaled = scaler.fit_transform(X_customer)
# X_customer_scaled is now a NumPy array
print("\nScaled features (first 5 rows):")
print(X_customer_scaled[:5])
```

Determine K (Elbow Method):

```
wcss = []
k_range = range(1, 11) # Test K from 1 to 10

for k_val in k_range:
 kmeans_temp = KMeans(n_clusters=k_val, init='k-means++',
random_state=42, n_init=10)  kmeans_temp.fit(X_customer_scaled)
 wcss.append(kmeans_temp.inertia_)

# Plot the Elbow
plt.figure(figsize=(10, 6))
plt.plot(k_range, wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Inertia)')
plt.xticks(k_range)
plt.grid(True)
plt.show()
# [Image: Matplotlib plot of WCSS vs. K, showing an "elbow" typically around
```

K=5 for this dataset] # Based on the plot, choose the K where the WCSS starts to decrease more slowly. # For the typical customer segmentation dataset, K=5 is often chosen.
optimal_k = 5 # Assume chosen from Elbow plot
print(f"\nOptimal K chosen (example): {optimal_k}")

Train K-Means Model with Optimal K:

kmeans_model = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42, n_init=10) kmeans_model.fit(X_customer_scaled) # Fit on scaled data
print("\nK-Means model trained with chosen K.")

Get Cluster Labels and Centroids:

cluster_labels = kmeans_model.labels_ # Cluster assignment for each data point

centroids_scaled = kmeans_model.cluster_centers_ # Coordinates of centroids (in scaled space)

Add cluster labels back to the original DataFrame (or a copy) for analysis
customer_df['Cluster'] = cluster_labels
print("\nCustomer data with cluster labels (head):")
print(customer_df.head())

print("\nScaled Centroids:")
print(centroids_scaled)

# To see centroids in original scale (if needed for interpretation)
centroids_original_scale = scaler.inverse_transform(centroids_scaled)
print("\nCentroids in Original Scale (AnnualIncome, SpendingScore):")
print(pd.DataFrame(centroids_original_scale, columns=['AnnualIncome_Centroid', 'SpendingScore_Centroid']))

Visualize Clusters: Scatter plot of the two features, colored by cluster.

plt.figure(figsize=(12, 8))
sns.scatterplot(x=X_customer_scaled[:, 0], y=X_customer_scaled[:, 1], hue=cluster_labels, palette=sns.color_palette('viridis', n_colors=optimal_k), s=100, alpha=0.7, legend='full') # Plot centroids
plt.scatter(centroids_scaled[:, 0], centroids_scaled[:, 1], marker='X', s=300, color='red', label='Centroids', edgecolor='black')

plt.title(f'Customer Segments (K={optimal_k}) - Scaled Data')
plt.xlabel('Annual Income (Scaled)')
plt.ylabel('Spending Score (Scaled)')

```
plt.legend()
plt.grid(True)
plt.show()
```

Exercise

1. Load the Customer_Segmentation.csv dataset.
2. Select only the Age and SpendingScore columns for clustering.
3. Scale these two selected features using StandardScaler.
4. Use the Elbow Method: Calculate and plot WCSS for K values from 1 to 10 using the scaled Age and SpendingScore data. Based on your plot, choose an appropriate value for K. Justify your choice 5. Train a K-Means model using your chosen K on the scaled Age and SpendingScore data. Use random_state=42 and n_init=10.
6. Assign the cluster labels back to your DataFrame containing Age and SpendingScore. 7. Create a scatter plot of Age (scaled) vs. SpendingScore (scaled), coloring the points by their assigned cluster label. Mark the cluster centroids on this plot.
8. Based on your plot and cluster assignments, briefly try to describe one or two of the clusters (e.g., "Cluster 0 seems to be younger customers with high spending scores").

Lab Session 13

Association Rule Mining: Apriori Algorithm

Objective

Apply the Apriori algorithm to find frequent itemsets and generate association rules from transactional data. Understand and interpret support, confidence, and lift.

Dataset

Market_Basket.csv (Conceptual - Instructor provides CSV). Each row represents a transaction. Items can be in a single comma-separated string column, or the data might be in a 'tidy' format (one row per item per transaction).

Association Rule Mining Basics:

- Goal: Discover relationships between items in large datasets (e.g., "customers who buy X also tend to buy Y").
- Itemset: A collection of one or more items.
- Support(Itemset): Proportion of transactions containing the itemset.

Support(X) = (No. of transactions containing X) / (Total transactions)

- Confidence(X -> Y): Likelihood of item Y being purchased when item

X is purchased. Confidence(X -> Y) = Support(X U Y) / Support(X)

- Lift(X -> Y): Measures how much more likely Y is purchased when X is purchased, compared to Y's overall purchase likelihood.

Lift(X -> Y) = Support(X U Y) / (Support(X) * Support(Y)).

  - o Lift = 1: X and Y are independent.
  - o Lift > 1: Positive correlation (presence of X increases likelihood of Y).
  - o Lift < 1: Negative correlation

Apriori Algorithm:

1. Set a minimum support threshold (min_sup).
2. Find all itemsets with support >= min_sup (frequent itemsets). Uses an iterative approach: find frequent 1- itemsets, then use them to generate candidate 2-itemsets, find frequent 2-itemsets, and so on (Apriori principle: if an itemset is frequent, all its subsets must also be frequent).
3. Generate association rules from frequent itemsets that meet a minimum confidence threshold (min_conf).

Data Preparation for mlxtend:

Requires a one-hot encoded Pandas DataFrame where:

- Each row is a transaction.
- Each column is an item.

- Values are True/False or 1/0 indicating item presence in the transaction.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Loading data from CSV:


# df_raw = pd.read_csv('Market_Basket.csv')
# transactions_list = df_raw['Items'].apply(lambda x: x.split(',')).tolist()

# Transform data into one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(transactions_list).transform(transactions_list)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)
print("One-Hot Encoded Transaction Data:")
print(df_onehot.head())
```

Finding Frequent Itemsets:

```
min_confidence_threshold = 0.6 # Example: rule must have at
least 60% confidence rules = association_rules(frequent_itemsets,
metric="confidence",
min_threshold=min_confidence_threshold)

# Sort rules by lift or confidence for better readability
rules_sorted = rules.sort_values(by=['lift', 'confidence'], ascending=[False, False])
print("\nGenerated Association Rules (confidence >= 0.6, sorted by
lift & confidence):") print(rules_sorted[['antecedents', 'consequents',
'support', 'confidence', 'lift']])
```

Interpreting Rules:

- Antecedents: {'itemA', 'itemB'}
- Consequents: {'itemC'}
- Rule: If a customer buys itemA and itemB, they are likely to buy itemC with X confidence and Y lift.

Exercise

1. Create a Python list of lists representing the following transactions:
   - o T1: Apple, Banana, Milk
   - o T2: Banana, Bread
   - o T3: Apple, Banana, Bread, Milk
   - o T4: Apple, Milk
   - o T5: Banana, Bread, Diapers
2. Use TransactionEncoder from mlxtend.preprocessing to transform this list into a one-hot encoded Pandas DataFrame.
3. Apply the apriori algorithm to this DataFrame to find frequent itemsets. Use a min_support of 0.4 (meaning an itemset must appear in at least 40% of the 5 transactions, so >= 2 transactions). Print the resulting DataFrame of frequent itemsets.
4. Generate association rules from these frequent itemsets using association_rules. Use a min_threshold of 0.7 for the "confidence" metric.

5. Print the generated rules, showing antecedents, consequents, support, confidence, and lift. 6. Identify and interpret one rule that has a lift greater than 1. (e.g., "The rule {Antecedent} => {Consequent} has a lift of L. This means customers buying {Antecedent} are L times more likely to buy {Consequent} than an average customer.")

DEPARTMENT OF SOFTWARE ENGINEERING

BACHELORS IN SOFTWARE ENGINEERING

Course Code: SE-407

Course Title: Data Warehouse & Mining

Complex Engineering Activity (CEA)

BE Batch 2022, Spring Semester 2025

Course Learning Outcome

CLO3: DEMONSTRATE the use of modern ETL tools for data warehouse development and use data mining algorithms to build analytical applications (P3-PLO5)

Complex problem-solving attributes (CPA) covered (as per PEC - OBA manual – 2019) •

CPA-1 Range of resources: Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies).

• **CPA-2 Level of interaction:** Require resolution of significant problems arising from interactions between wide-ranging or conflicting technical, engineering or other issues.

Problem Statement

Objective: The primary objective of this Complex Engineering Activity is to provide students with hands-on experience in the complete lifecycle of a data mining project. This includes sourcing and preparing a real-world dataset, defining a relevant data mining problem, selecting and applying appropriate data mining algorithms, implementing these algorithms using modern tools, performing a rigorous comparative analysis of the results, and demonstrating their solution and insights.

Project Task:

Students are required to perform the following tasks:

1. **Dataset Sourcing and Justification:**
   - o **Identify and select a suitable real-world dataset from publicly available data repositories (e.g., Kaggle, UCI Machine Learning Repository, Data.gov, Awesome Public Datasets).**
   - o **Provide a detailed justification for the choice of the dataset, explaining its relevance, potential for interesting insights, and suitability for data mining tasks.**
   - o **Describe the dataset's characteristics (e.g., number of instances, attributes, data types, potential challenges like missing values).**
2. **Problem Definition:**
   - o **Based on the selected dataset, clearly define a specific and relevant data mining problem. This could be:**

▪ **Classification: Predicting a categorical class label (e.g., customer churn, disease** diagnosis).

▪ Clustering: Grouping similar data instances together (e.g., customer segmentation, anomaly detection).

▪ Association Rule Mining: Discovering interesting relationships between items (e.g., market basket analysis).

▪ Regression: Predicting a continuous numerical value (e.g., house price prediction, stock price forecasting).

o Explain why this problem is significant and what insights might be gained from solving it. 3. Algorithm Selection and Application:

o Select at least two different data mining algorithms that are appropriate for addressing the defined problem using the chosen dataset.

o Provide a clear rationale for selecting these algorithms, discussing their underlying principles, strengths, weaknesses, and suitability for the dataset and problem.

o Perform necessary data preprocessing steps (e.g., data cleaning, transformation, feature selection/engineering).

4. Implementation and Execution:

o Implement the chosen algorithms using appropriate tools or programming languages (e.g., Python with libraries like scikit-learn, Pandas, NumPy; R; Weka).

o Ensure the implementation is well-documented within the code.

o Execute the algorithms on the prepared dataset.

5. Results and Comparative Analysis:

o Prepare to present the results obtained from each algorithm clearly and concisely (e.g., using tables, charts, confusion matrices, cluster visualizations during the demonstration).

o Perform a thorough comparative analysis of the applied algorithms. This analysis should include:

▪ Relevant performance metrics (e.g., accuracy, precision, recall, F1-score for classification; silhouette score, Davies-Bouldin index for clustering; support, confidence, lift for association rules; Mean Squared Error, R-squared for regression).

▪ Discussion of the strengths and weaknesses of each algorithm in the context of the specific problem and dataset.

▪ Insights gained from the results of each algorithm and from the comparison.

6. Conclusion and Reflection:

o Draw overall conclusions based on the analysis.

o Summarize the key findings and their implications.

o Reflect on the challenges faced during the project (e.g., data quality issues, algorithm complexity, interpretation of results).

o Discuss the learning outcomes and any new skills or knowledge acquired.

7. Demonstration:

o Prepare and deliver a demonstration of the implemented data mining solution.

o The demonstration should cover the dataset, problem definition, algorithms used, implementation details (walkthrough of key code sections), results, comparative analysis, and conclusions.

o Be prepared to answer questions about the project.

Constraints/ Assumptions

• Students must use publicly available datasets; proprietary or private data is not permitted without explicit instructor approval and ethical clearance.

• The chosen dataset should be of sufficient complexity to warrant a data mining approach but manageable within the project timeframe.

• Students are expected to work individually or in groups (as specified by the instructor). • All sources of information and code (if adapted from external sources) must be properly cited.

• The focus is on the application and comparative analysis of algorithms; developing entirely new algorithms is not required.

Identification of areas where the use of computational/ modern tool is required.

The completion of this CEA necessitates the use of modern computational tools and programming environments. Key areas include:

• Data Acquisition and Preprocessing: Tools for downloading, cleaning, transforming, and preparing datasets (e.g., Python with Pandas and NumPy, R).

• Algorithm Implementation: Data mining libraries and software (e.g., scikit-learn in Python, R packages, Weka).

• Data Visualization (for demonstration): Libraries for creating charts and graphs to present results and insights (e.g., Matplotlib, Seaborn in Python; ggplot2 in R).

• Presentation/Demonstration Tools: Software for presenting findings (e.g., PowerPoint, Google Slides, Jupyter Notebooks for live demos).

Expected outcomes

Upon successful completion of this CEA, students are expected to deliver/demonstrate:

• Upon successful completion of this CEA, students are expected to deliver/demonstrate: o A well-chosen and justified

real-world dataset.

o **A clearly defined data mining problem relevant to the chosen dataset.**

o **A functional implementation of at least two appropriate data mining algorithms.**

o **A clear and comprehensive demonstration of their project, including:**

- **Dataset description and problem definition.**
- **Algorithm selection rationale and key implementation details.**
- **Clear presentation of results using appropriate metrics and visualizations.**
- **A thorough comparative analysis of the algorithms.**
- **Meaningful conclusions and reflections on the project.**

• **Demonstrated ability to apply data mining techniques to solve a practical problem and communicate the process and findings effectively.**

• **Enhanced understanding of the strengths, weaknesses, and applicability of different data mining algorithms.** • **Improved skills in using data mining tools and software for implementation and demonstration.**

Instructor's Name & Signature: Dr. Mustafa Latif____________

Date: ___________________________

Semester: _Spring 2025__________

Batch: ___2022__ ____________

F/OBEM 01/18/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining
Assessment Rubric for Complex Engineering Activity (CEA)

Criterion	**Level of Attainment**		
	0-1	1.5	2
Dataset Sourcing & Justification	Dataset is inappropriate, poorly described, or justification is missing/weak.	Dataset is adequately selected, with some description and basic justification.	Dataset is well-chosen, clearly described, with good justification for its relevance and suitability.
Problem Definition	Problem is ill-defined, irrelevant to the dataset, or not a data mining problem.	Problem is defined but may lack clarity, relevance, or specificity.	Problem is clearly defined, relevant to the dataset, and appropriate as a data mining task.
Algorithm Selection & Justification	Algorithms are inappropriate for the problem/dataset, or justification is absent/flawed.	At least two algorithms are selected, with some attempt at justification, but choices may not be optimal.	Appropriate algorithms (at least two) are selected with clear and logical justification for their suitability.
Implementation & Execution Quality	Implementation is non functional, incomplete, or uses tools incorrectly. Significant errors.	Implementation is partially functional, with some errors or incorrect use of tools/parameters.	Implementation is mostly correct and functional, using appropriate tools and parameters with minor issues. Well commented code.
Demonstration: Results, Analysis, Clarity & Communication	Demonstration is unclear, poorly organized. Results are missing/incorrect. Analysis is superficial or absent. Poor communication.	Demonstration shows basic understanding. Results are presented but may lack clarity. Analysis is basic. Communication is adequate.	Demonstration is clear and well-organized. Results are well-presented with good analysis. Good communication skills.

Student’s Name: ____ Roll No.: ____________

Total Score = ___________________________

Instructor's Signature: ____________________

F/OBEM 01/05/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining

Software Use Rubric				
Criterion	Extent of Achievement			
	0	2	4	5
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
How well is the code documented?	The code has no comments or documentation.	The code is partially documented, with comments explaining more than half but not all of the main functions and logic.	The code is mostly documented, covering most functions and logic but missing some details.	The code is fully documented, with comprehensive comments explaining all functions, logic, and purpose throughout.

Laboratory Session No. Date:

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

F/OBEM 01/05/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining

Software Use Rubric				
Criterion	Extent of Achievement			
	0	2	4	5
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
How well is the code documented?	The code has no comments or documentation.	The code is partially documented, with comments explaining more than half but not all of the main functions and logic.	The code is mostly documented, covering most functions and logic but missing some details.	The code is fully documented, with comprehensive comments explaining all functions, logic, and purpose throughout.

Laboratory Session No. Date:

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

F/OBEM 01/05/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining

Software Use Rubric				
Criterion	Extent of Achievement			
	0	2	4	5
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
How well is the code documented?	The code has no comments or documentation.	The code is partially documented, with comments explaining more than half but not all of the main functions and logic.	The code is mostly documented, covering most functions and logic but missing some details.	The code is fully documented, with comprehensive comments explaining all functions, logic, and purpose throughout.

Laboratory Session No. ______________ Date: ________________________

Weighted CLO Score	
Remarks	

Instructor's Signature with Date	

F/OBEM 01/18/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining
Assessment Rubric for Open Ended Lab

Criterion	**Level of Attainment**	
	0	1
Data Preparation & Model Training	Fails to correctly use/reference training/testing sets from Lab 10, OR GaussianNB model is not trained correctly or not trained at all.	Correctly uses existing training/testing sets (X_train_r, y_train_r, X_test_r, y_test_r) and successfully trains the GaussianNB model.
Prediction with Naive Bayes Model	Predictions on the test data are not made, or are made incorrectly.	Successfully makes predictions on the corresponding test data using the trained Naive Bayes model.
Naive Bayes Model Evaluation	Fails to calculate/print one or more evaluation metrics (accuracy, confusion matrix, classification report), OR metrics are incorrect.	Accurately calculates and prints the accuracy, confusion matrix, and classification report for the Naive Bayes model.
Comparison with Decision Tree Model	Fails to compare the Naive Bayes accuracy with the Decision Tree accuracy from Lab 10, OR comparison is missing key elements.	Clearly compares the overall accuracy of the GaussianNB model (for PassedReading) with the Decision Tree model from Lab 10.
Analysis of Precision and Recall	Fails to identify which model gives better precision/recall for Class '1', OR commentary is missing or significantly flawed.	Correctly identifies which model (Naive Bayes vs. Decision Tree) gives better precision and recall for Class '1' (Passed) and provides a brief, meaningful comment.

Student's Name: ____ Roll No.: ____________

Total Score = ___________________________

Instructor's Signature: ____________________

F/OBEM 01/18/00

NED University of Engineering & Technology
Department of Software Engineering
Course Code & Title: SE-407 Data Warehouse & Mining
Performance based Rubric Evaluation (Final Lab Exam)

Software Use Rubric		
Criterion	Extent of Achievement	
	0	1
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered all the questions.
How well is the code documented?	The code has no comments or documentation.	The code is fully documented, with comprehensive comments explaining all functions, logic, and purpose throughout.

Student's Name: ____ Roll No.: _____________

Total Score = _____________________________

Instructor's Signature: _____________________