ETL Process for SAP Data

1. Introduction

The Extract, Transform, Load (ETL) process is a crucial component of data management, enabling businesses to process and analyze large datasets efficiently. SAP ERP systems generate vast amounts of transactional data, including orders, customer details, shipments, and financial records. However, raw data in SAP is often complex and needs processing before meaningful insights can be extracted.

A well-defined ETL pipeline streamlines data flow from source systems to analytical platforms, ensuring accuracy and reliability. By implementing automation, businesses can minimize manual intervention, reduce human errors, and improve processing speed. Additionally, structured data transformation ensures that various business units can access consistent and meaningful insights.

In this project, we implement an ETL pipeline using Python to extract SAP data, transform it into a structured format, and load it into a MySQL database. The final dataset is then analyzed and visualized using **Power BI**, enabling better decision-making based on real-time business insights. The project is designed to automate data handling, improve data quality, and provide analytical capabilities to businesses.

Furthermore, integrating visualization tools like Power BI enhances data-driven decision-making by providing interactive dashboards. These dashboards allow stakeholders to monitor key performance indicators, track trends, and gain actionable insights, ultimately improving operational efficiency and strategic planning.

2. Problem Statement

Organizations using SAP ERP systems often face challenges in **data extraction, transformation, and integration** into analytical platforms. The primary challenges include:

- Extracting relevant data from SAP tables while ensuring completeness and accuracy.
- Cleaning, structuring, and transforming raw data to make it usable for analysis.
- Loading the processed data into a database for structured querying.
- Creating an interactive **Power BI dashboard** to visualize sales trends, shipments, and customer insights effectively.
- Reducing manual efforts in handling data and improving real-time reporting efficiency.

Businesses rely on accurate and timely data to make informed decisions. However, extracting SAP data manually can be time-consuming and prone to inconsistencies. Data stored in unstructured formats can hinder data analysis, making it difficult to derive meaningful insights.

Additionally, improper data transformation can lead to inaccurate reports, affecting business strategies.

A structured ETL process ensures that **business users** can access clean, reliable, and real-time data for **better decision-making and performance tracking**. By automating the process, organizations can reduce data processing time, improve accuracy, and enhance reporting capabilities.

Finally, a well-structured ETL pipeline helps organizations overcome data integration challenges by ensuring seamless connectivity between SAP ERP systems, databases, and visualization tools. This allows for a smooth flow of data across different business functions, facilitating better collaboration and efficiency.

3. Objectives

The key objectives of this ETL implementation are:

- **Automate** the extraction of SAP data from structured files (CSV/Excel) for further processing.
- **Perform data validation and transformation** to ensure completeness, consistency, and accuracy.
- **Load the cleaned data into MySQL**, ensuring optimized query performance and structured storage.
- **Use Power BI for visualization**, allowing stakeholders to analyze sales, customer behavior, and delivery performance in real time.
- **Develop a user-friendly interface** to manage ETL execution seamlessly.
- **Reduce errors and inconsistencies** in SAP data processing through automated transformations.
- **Enable real-time data updates** to ensure up-to-date information for reporting and analysis.

A key objective of this project is to ensure that businesses can manage their SAP data efficiently without requiring advanced technical expertise. By providing a user-friendly interface, users can trigger ETL operations with minimal effort, reducing dependency on IT teams. The project also aims to make data-driven insights accessible to all business stakeholders.

Additionally, this ETL solution is designed to be scalable and adaptable to different SAP data formats and structures. It ensures that as business requirements evolve, the pipeline can accommodate new data sources and transformation rules without significant rework.

4. Technologies Used

To implement this ETL process, we use the following technologies:

- **Python**: Core programming language for scripting the ETL pipeline.
- **Pandas**: Used for data transformation, cleaning, and preprocessing.
- **MySQL**: Acts as the data warehouse for structured storage and efficient querying.
- **SQLAlchemy**: Provides an abstraction for connecting Python with MySQL databases and executing database operations efficiently.
- **Tkinter**: GUI toolkit for creating an interactive interface to manage the ETL process.
- **Power BI**: Used for data visualization, enabling stakeholders to explore business trends and generate reports.
- **OS Module**: Handles file and folder creation dynamically.
- **pymysql**: Enables seamless interaction between Python and MySQL.

The combination of these technologies ensures a reliable and efficient ETL pipeline. Python provides flexibility in data processing, MySQL offers a structured database for querying, and Power BI enables comprehensive data visualization. By integrating these technologies, businesses can achieve a streamlined data management process.

Furthermore, these technologies have been selected to ensure scalability and ease of use. Python libraries like Pandas simplify data handling, while SQLAlchemy and pymysql facilitate smooth database interactions. The use of a GUI enhances accessibility, making it easier for non-technical users to run the ETL process.

5. ETL Process Breakdown

The ETL pipeline is divided into three key stages:

5.1 Extract

The extraction step reads SAP data from CSV/Excel files, which represent different SAP tables such as Orders, Customers, and Shipments. The extracted data is stored in a structured format for further processing. Automated extraction ensures data accuracy and eliminates manual errors.

```
import pandas as pd

import os



def extract_data(file_path, sheet_name=None):

    """
```

```
    Extracts data from an Excel/CSV file and loads it into a DataFrame.
    If an Excel sheet name is provided, it reads the specific sheet.
    """
    print(f"Starting extraction from {file_path}...")

    if file_path.endswith('.xlsx'):
        df = pd.read_excel(file_path, sheet_name=sheet_name)
    elif file_path.endswith('.csv'):
        df = pd.read_csv(file_path)
    else:
        raise ValueError("Unsupported file format. Please provide an Excel or CSV file.")

     print(f"Successfully extracted {len(df)} records from {sheet_name if sheet_name else 'CSV File'}")
    return df

if __name__ == "__main__":
    # File path
    excel_file_path = "SAP-DataSet.xlsx"
    csv_file_path = ""  # Optional CSV file

    # Sheets to extract from Excel
    sheets = ['KNA1', 'VBAK', 'VBAP', 'LIKP', 'LIPS', 'VTTK', 'VTTP']

    # Extract data from Excel
```

```
    excel_dataframes = {sheet: extract_data(excel_file_path, sheet) for sheet in sheets}


    # Extract data from CSV (if needed)
    csv_data = extract_data(csv_file_path) if os.path.exists(csv_file_path) else None


    # Save extracted data for transformation step
    for sheet, df in excel_dataframes.items():
        df.to_csv(f"{sheet}.csv", index=False)


    if csv_data is not None:
        csv_data.to_csv("extracted_data.csv", index=False)


    print("Extraction process completed. Data saved successfully!")
```

5.2 Transform

During transformation, the extracted data undergoes validation, cleaning, and restructuring. This step removes duplicates, handles missing values, and maps SAP-specific field names to more user-friendly ones. Additionally, column standardization and data type corrections are applied to ensure data consistency.

```
import pandas as pd


def clean_data(df, table_name):
    """
    Cleans and transforms the extracted data.
```

```
    """
    print(f"Starting data cleaning for {table_name}...")

    # Drop duplicates
    df = df.drop_duplicates()

    # Fill missing values with appropriate defaults
    df = df.fillna('Unknown')

    # Ensure date format consistency
    date_columns = [col for col in df.columns if 'date' in col.lower()]
    for col in date_columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')

    # Strip leading/trailing spaces from string columns
    str_columns = df.select_dtypes(include=['object']).columns
    df[str_columns] = df[str_columns].apply(lambda x: x.str.strip())

    print(f"{table_name} data cleaned successfully.")
    return df

if __name__ == "__main__":
    # Load extracted CSV files
    tables = ['KNA1', 'VBAK', 'VBAP', 'LIKP', 'LIPS', 'VTTK', 'VTTP']
    cleaned_data = {}
```

```
    for table in tables:
        print(f"Processing transformation for {table}...")
        df = pd.read_csv(f"{table}.csv")
        cleaned_data[table] = clean_data(df, table)
        print(f"Transformation complete for {table}.")

    # Save transformed data as CSV and Excel
    writer = pd.ExcelWriter("transformed_data.xlsx", engine='xlsxwriter')

    for table, df in cleaned_data.items():
        df.to_csv(f"cleaned_{table}.csv", index=False)
        df.to_excel(writer, sheet_name=table, index=False)
        print(f"Cleaned data saved for {table}. Ready for loading.")

    writer.close()

    print("All transformations are complete! Ready for data loading.")
```

5.3 Load

The transformed data is loaded into a **MySQL database**, making it accessible for analysis and reporting.

```
import pandas as pd
from sqlalchemy import create_engine
```

```
# Database connection details
DB_USER = "root"
DB_PASSWORD = ""
DB_HOST = "127.0.0.1"
DB_NAME = "sap_data_warehouse"

# Create a database connection
engine = create_engine(f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/{DB_NAME}")

def load_data(file_path, table_name):
    """
    Loads cleaned data into MySQL database.
    """
        df = pd.read_csv(file_path) if file_path.endswith('.csv') else pd.read_excel(file_path)
    print(f"Starting to load {len(df)} records into {table_name}...")
    df.to_sql(table_name, con=engine, if_exists='replace', index=False)
    print(f"{table_name} loaded successfully into the database.")

if __name__ == "__main__":
    # Tables to load
    tables = ['KNA1', 'VBAK', 'VBAP', 'LIKP', 'LIPS', 'VTTK', 'VTTP']
```

```
    for table in tables:

        print(f"Initiating load process for {table}...")

        load_data(f"cleaned_{table}.csv", table)

        print(f"Data successfully loaded for {table}.")



    print("All data loading processes are complete! Database is ready.")
```

The full ETL process ensures seamless data management, reducing redundancy and improving data reliability. The combination of automation and visualization enables businesses to maximize the value of their SAP data.

5.4 Interface for ETL Process

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import subprocess
import os
import time

def upload_file():
    global file_path
    file_path = filedialog.askopenfilename(filetypes=[("", "*.csv"), ("",
"*.xlsx")])
    if file_path:
        upload_label.config(text=f"File Selected:
{os.path.basename(file_path)}", fg="#2ecc71")
    else:
        messagebox.showerror("Error", "No file selected. Please select a
valid file.")

def update_progress(progress_bar, percentage_label, value):
    progress_bar['value'] = value
    percentage_label.config(text=f"{value}% Completed", font=("Arial", 12,
"bold"))
    root.update_idletasks()
    time.sleep(0.5)
```

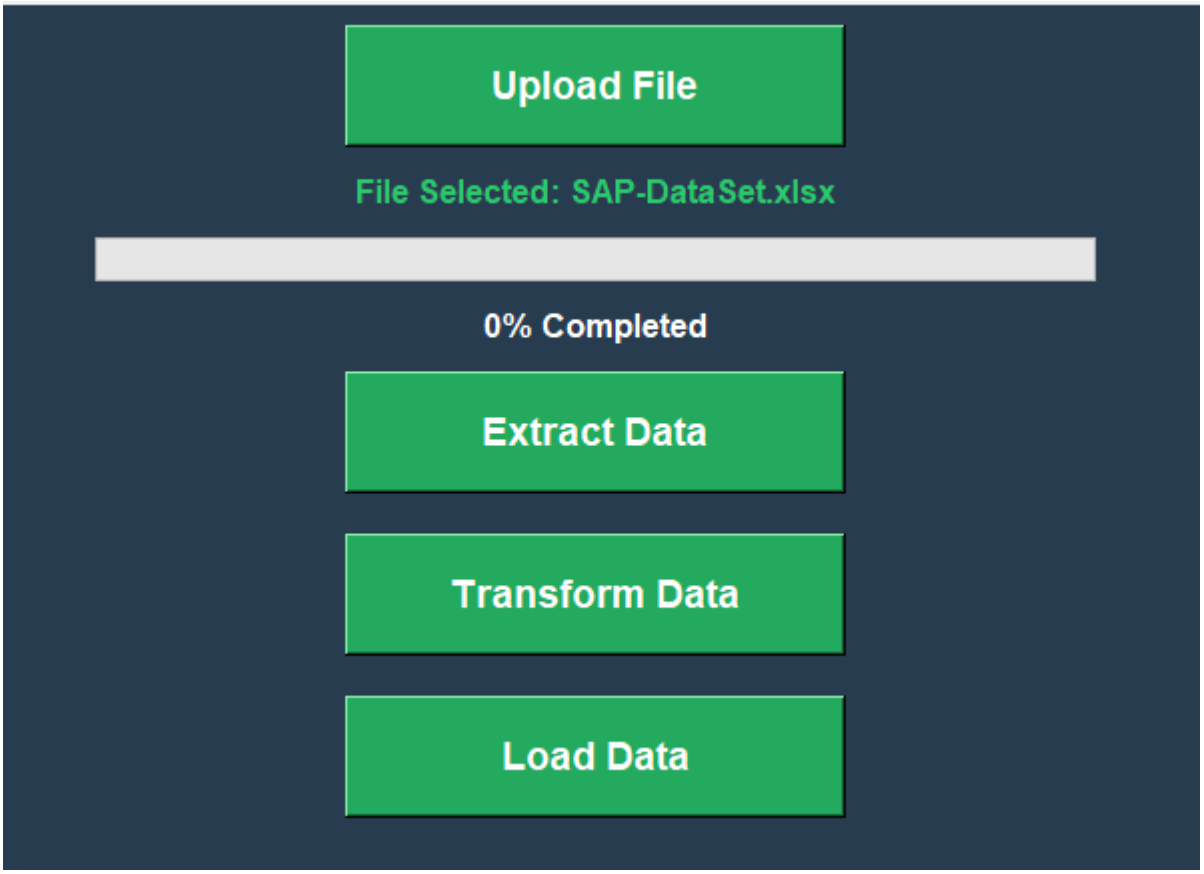
```
def run_script(script_name):
    try:
        subprocess.run(["python", script_name], check=True)
        return True
    except subprocess.CalledProcessError as e:
        messagebox.showerror("Error", f"Failed to execute {script_name}:
{str(e)}")
        return False

def extract_data():
    if 'file_path' in globals() and file_path:
        progress_bar['value'] = 0
        update_progress(progress_bar, percentage_label, 25)

        extract_success = run_script("extract.py")

        if extract_success:
            update_progress(progress_bar, percentage_label, 100)
            messagebox.showinfo("Success, Data extracted successfully!")
    else:
        messagebox.showerror("Error", "Please upload a file first")

def transform_data():
    progress_bar['value'] = 0
    update_progress(progress_bar, percentage_label, 25)

    transform_success = run_script("transform.py")

    if transform_success:
        update_progress(progress_bar, percentage_label, 100)
        messagebox.showinfo("Success", "Data transformed successfully!")

def load_data():
    progress_bar['value'] = 0
    update_progress(progress_bar, percentage_label, 25)

    load_success = run_script("load_to_sql.py")

    if load_success:
        update_progress(progress_bar, percentage_label, 100)
        messagebox.showinfo("Success", "Data loaded successfully into the
database!")
```

```
# Creating the GUI window
root = tk.Tk()
root.title("ETL Process Interface")
root.state("zoomed")  # Full-screen mode
root.configure(bg="#2c3e50")

# Styling
button_style = {"font": ("Arial", 14, "bold"), "bg": "#27ae60", "fg":
"white", "width": 20, "height": 2}
label_style = {"font": ("Arial", 12, "bold"), "bg": "#2c3e50", "fg":
"white"}

# Main Frame
main_frame = tk.Frame(root, bg="#2c3e50")
main_frame.pack(expand=True, fill="both")

# Upload File Section
upload_btn = tk.Button(main_frame, text="Upload File",
command=upload_file, **button_style)
upload_btn.pack(pady=10)

upload_label = tk.Label(main_frame, text="No file selected",
**label_style)
upload_label.pack()

# Progress Bar & Percentage Label
progress_bar = ttk.Progressbar(main_frame, orient="horizontal",
length=500, mode="determinate")
progress_bar.pack(pady=10)

percentage_label = tk.Label(main_frame, text="0% Completed",
**label_style)
percentage_label.pack()

# ETL Buttons
extract_btn = tk.Button(main_frame, text="Extract Data",
command=extract_data, **button_style)
extract_btn.pack(pady=10)

transform_btn = tk.Button(main_frame, text="Transform Data",
command=transform_data, **button_style)
transform_btn.pack(pady=10)
```

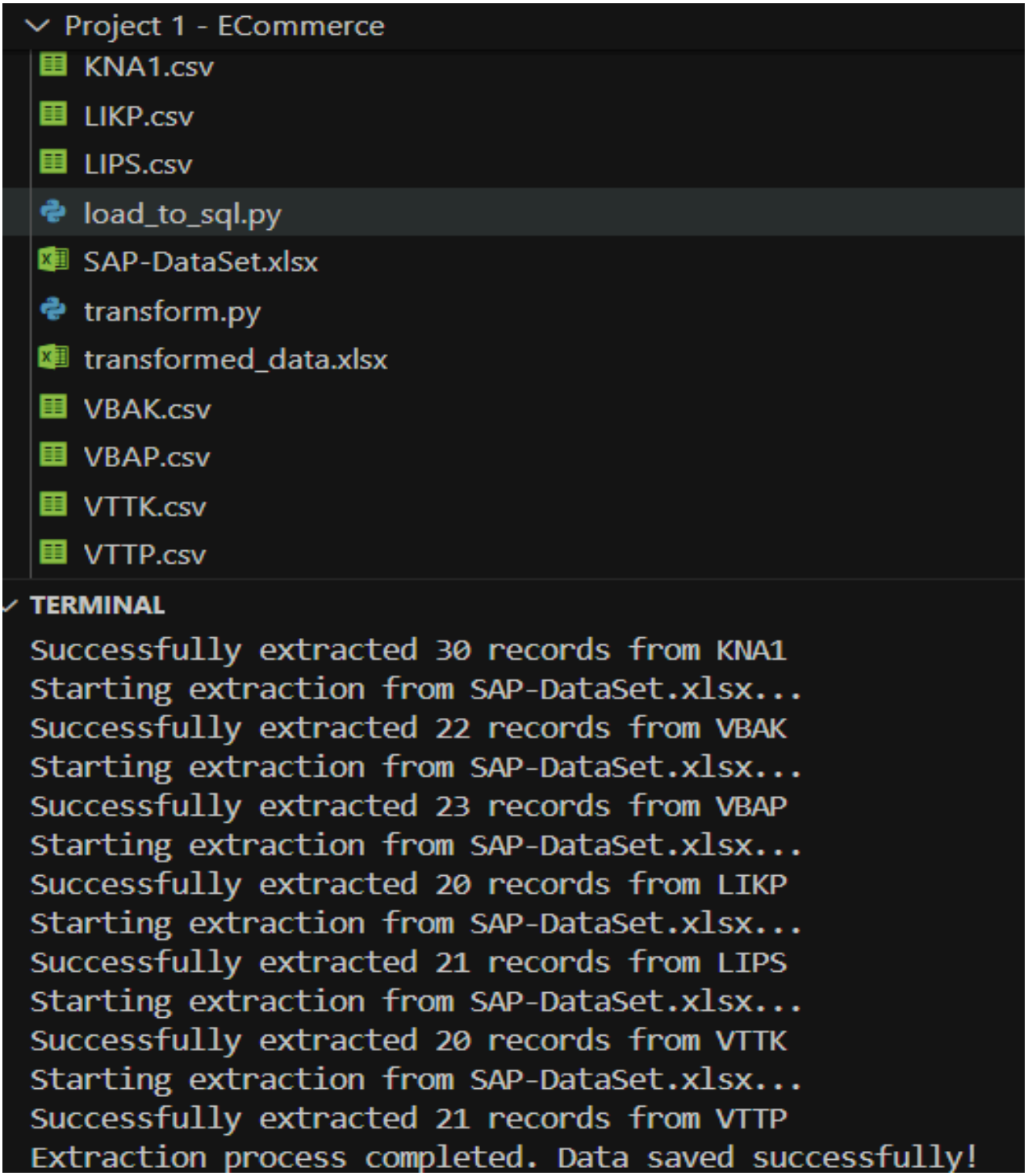
```
load_btn = tk.Button(main_frame, text="Load Data", command=load_data,
**button_style)
load_btn.pack(pady=10)

# Run the GUI
root.mainloop()
```

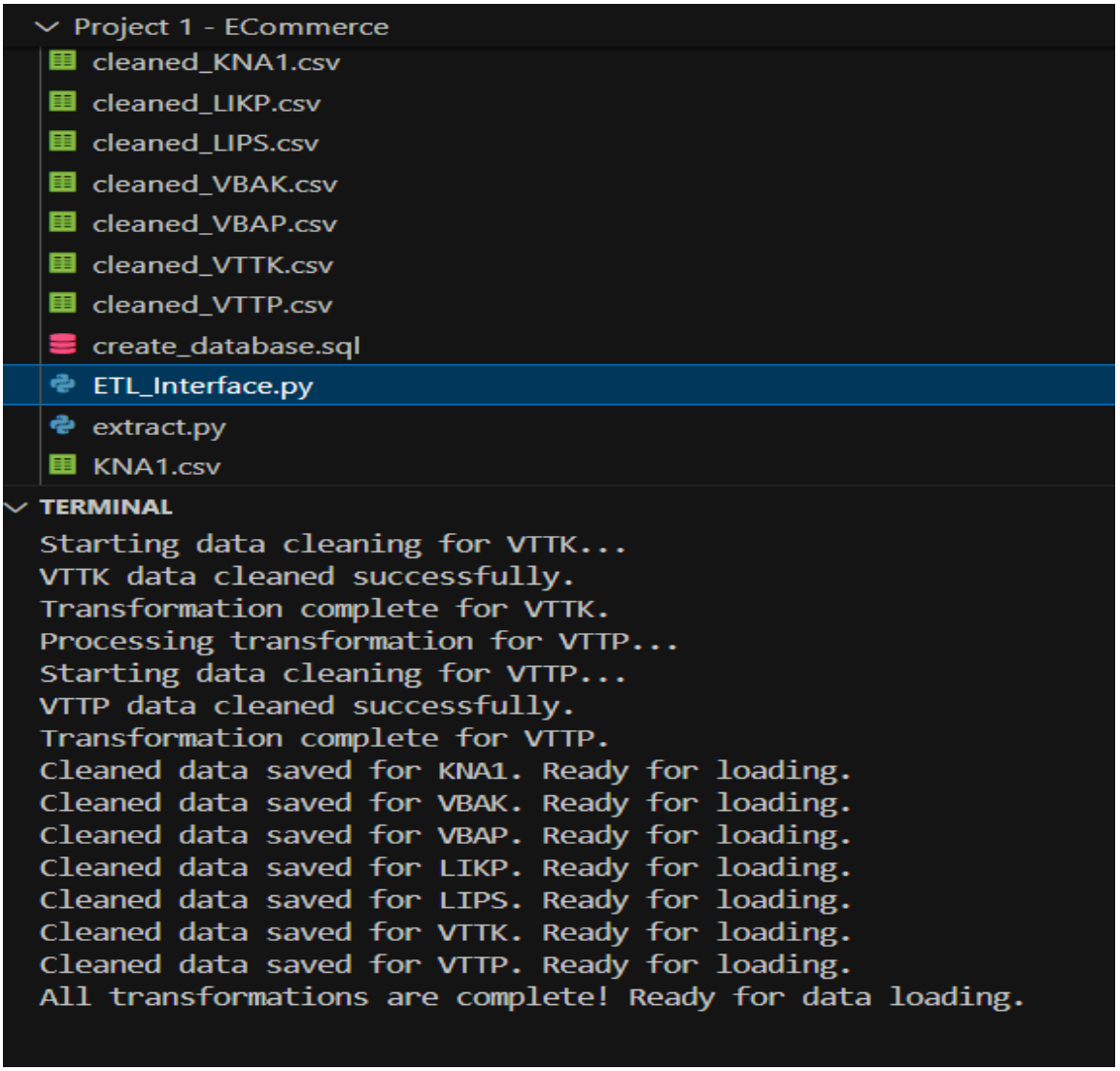
6. Results

6.1 Interface to Upload CSV/EXCEL file:



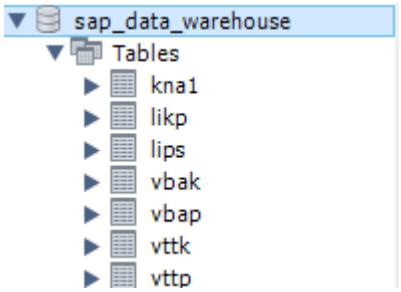
6.2Extract-Transform-LoadToSQLDataBase

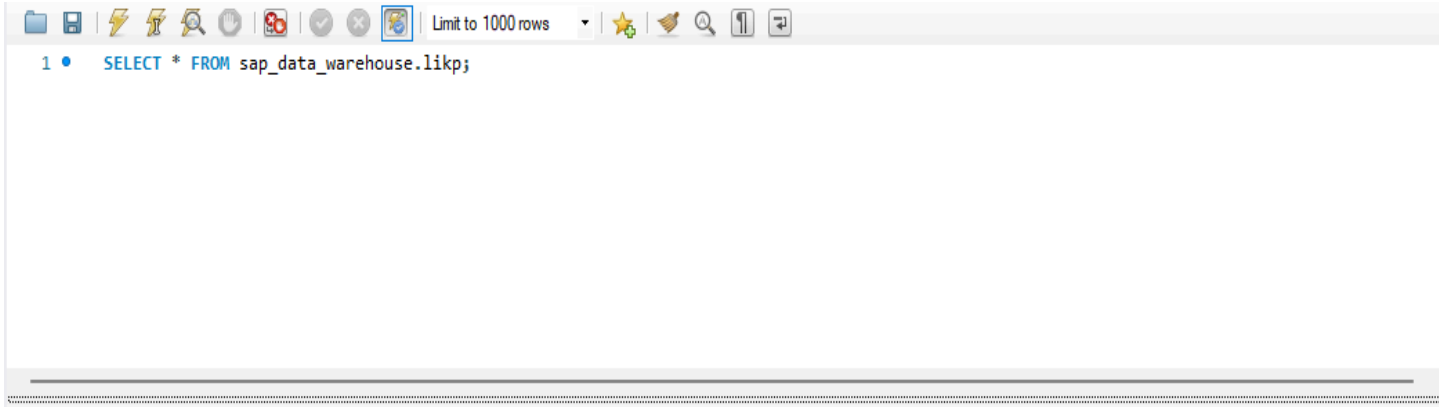





```
  df.to_sql(table_name, con=engine, if_exists='replace', index=False)
LIPS loaded successfully into the database.
Data successfully loaded for LIPS.
Initiating load process for VTTK...
Starting to load 20 records into VTTK...
C:\Users\satya\OneDrive\Desktop\ADMM\ADMM Projects\Project 1 - ECommerce\load_to_sql.py:19: UserWarning: The provided table n
ame 'VTTK' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Con
sider using lower case table names.
  df.to_sql(table_name, con=engine, if_exists='replace', index=False)
VTTK loaded successfully into the database.
Data successfully loaded for VTTK.
Initiating load process for VTTP...
Starting to load 21 records into VTTP...
C:\Users\satya\OneDrive\Desktop\ADMM\ADMM Projects\Project 1 - ECommerce\load_to_sql.py:19: UserWarning: The provided table n
ame 'VTTP' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Con
sider using lower case table names.
  df.to_sql(table_name, con=engine, if_exists='replace', index=False)
VTTP loaded successfully into the database.
Data successfully loaded for VTTP.
All data loading processes are complete! Database is ready.
```

6.3 DataBase mySQL Workbench [example with on table]





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

Delivery Number	Delivery Date	Sales Document	Shipping Point	Shipping Type	Delivery Status	Shipping Status	Route	Delivery Priority	Customer ID
1001001	2025-02-10	1000001	SP001	Standard	Open	Not Shipped	1	High	CUST001
1001002	2025-02-11	1000002	SP002	Express	Delivered	Shipped	2	Medium	CUST002
1001003	2025-02-12	1000003	SP001	Standard	Open	Not Shipped	3	Low	CUST003
1001004	2025-02-13	1000004	SP003	Express	Delivered	Shipped	4	High	CUST004
1001005	2025-02-14	1000005	SP001	Standard	Open	Not Shipped	5	Medium	CUST005
1001006	2025-02-15	1000006	SP002	Express	Delivered	Shipped	6	High	CUST006
1001007	2025-02-16	1000007	SP003	Standard	Open	Not Shipped	7	Low	CUST007
1001008	2025-02-17	1000008	SP001	Express	Delivered	Shipped	8	Medium	CUST008
1001009	2025-02-18	1000009	SP002	Standard	Open	Not Shipped	9	High	CUST009
1001010	2025-02-19	1000010	SP003	Express	Delivered	Shipped	10	Low	CUST010
1001011	2025-02-20	1000011	SP001	Standard	Open	Not Shipped	11	Medium	CUST011
1001012	2025-02-21	1000012	SP002	Express	Delivered	Shipped	12	High	CUST012
1001013	2025-02-22	1000013	SP003	Standard	Open	Not Shipped	13	Low	CUST013
1001014	2025-02-23	1000014	SP001	Express	Delivered	Shipped	14	Medium	CUST014
1001015	2025-02-24	1000015	SP002	Standard	Open	Not Shipped	15	High	CUST015
1001016	2025-02-25	1000016	SP003	Express	Delivered	Shipped	16	Low	CUST016
1001017	2025-02-26	1000017	SP001	Standard	Open	Not Shipped	17	Medium	CUST017
1001018	2025-02-27	1000018	SP002	Express	Delivered	Shipped	18	High	CUST018
1001019	2025-02-28	1000019	SP003	Standard	Open	Not Shipped	19	Low	CUST019
1001020	2025-03-01	1000020	SP001	Express	Delivered	Shipped	20	Medium	CUST020

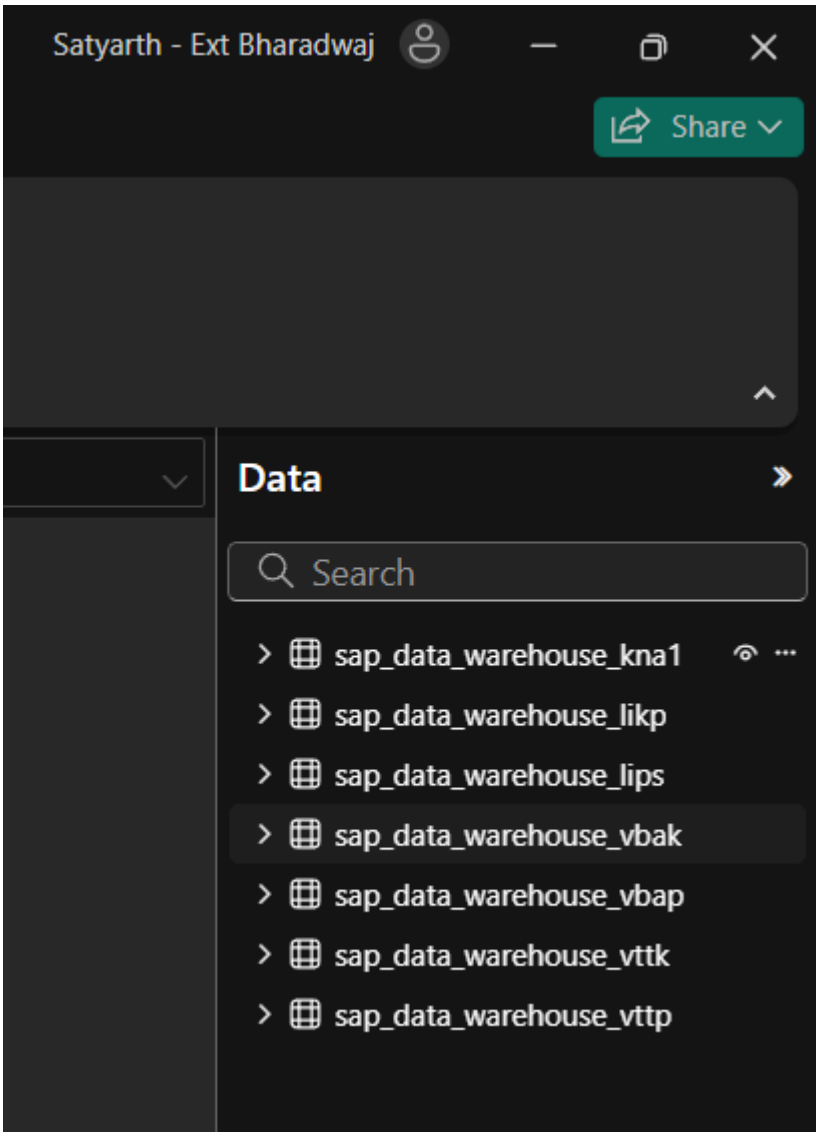
likp 1 ×

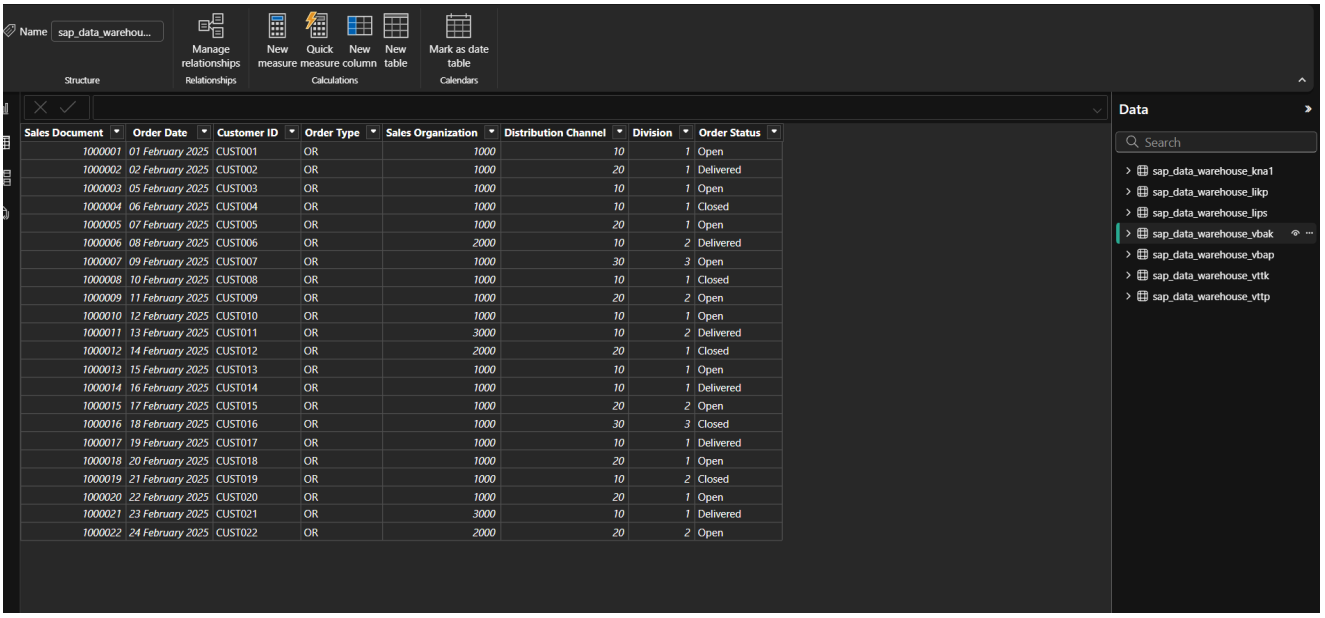
Output

Action Output

#	Time	Action	Message
1	11:29:55	DROP DATABASE `sap_data_warehouse`	7 row(s) affected
2	11:33:40	Create Database if not exists sap_data_warehouse	1 row(s) affected
3	11:34:30	SELECT * FROM sap_data_warehouse.kna1 LIMIT 0, 1000	30 row(s) returned
4	12:39:02	DROP DATABASE `sap_data_warehouse`	7 row(s) affected
5	12:39:05	Create Database if not exists sap_data_warehouse	1 row(s) affected
6	13:05:15	SELECT * FROM sap_data_warehouse.likp LIMIT 0, 1000	20 row(s) returned

6.4 PowerBI Tables [After Connecting to mySQL Datatbase]





Sales Document	Order Date	Customer ID	Order Type	Sales Organization	Distribution Channel	Division	Order Status
1000001	01 February 2025	CUST001	OR	1000	10	1	Open
1000002	02 February 2025	CUST002	OR	1000	20	1	Delivered
1000003	05 February 2025	CUST003	OR	1000	10	1	Open
1000004	06 February 2025	CUST004	OR	1000	10	1	Closed
1000005	07 February 2025	CUST005	OR	1000	20	1	Open
1000006	08 February 2025	CUST006	OR	2000	10	2	Delivered
1000007	09 February 2025	CUST007	OR	1000	30	3	Open
1000008	10 February 2025	CUST008	OR	1000	10	1	Closed
1000009	11 February 2025	CUST009	OR	1000	20	2	Open
1000010	12 February 2025	CUST010	OR	1000	10	1	Open
1000011	13 February 2025	CUST011	OR	3000	10	2	Delivered
1000012	14 February 2025	CUST012	OR	2000	20	1	Closed
1000013	15 February 2025	CUST013	OR	1000	10	1	Open
1000014	16 February 2025	CUST014	OR	1000	10	1	Delivered
1000015	17 February 2025	CUST015	OR	1000	20	2	Open
1000016	18 February 2025	CUST016	OR	1000	30	3	Closed
1000017	19 February 2025	CUST017	OR	1000	10	1	Delivered
1000018	20 February 2025	CUST018	OR	1000	20	1	Open
1000019	21 February 2025	CUST019	OR	1000	10	2	Closed
1000020	22 February 2025	CUST020	OR	1000	20	1	Open
1000021	23 February 2025	CUST021	OR	3000	10	1	Delivered
1000022	24 February 2025	CUST022	OR	2000	20	2	Open

6.5 PowerBI Model View

6.6 PowerBI Dashboard Representation