High-Level ETL (Extract - Transform - Load) Flow

Goal: By the end of this tutorial, you will be able to

- Extract: Download a file from AWS S3 using Python's boto3.
- Transform: Clean, filter, or manipulate data in Python (often using libraries like pandas).
- Load: Insert the transformed data into a relational database via SQL statements.

Lab Assignment

1. Implement the following functions
   - `extract_from_csv(file_to_process: str) -> pd.DataFrame` : read the .csv file and return dataframe
   - `extract_from_json(file_to_process: str) -> pd.DataFrame` : read the .json file and return dataframe
   - `extract() -> pd.DataFrame` : extract data of heterogeneous format and combine them into a single dataframe.
   - `transform(df) -> pd.DataFrame` : function for data cleaning and manipulation.
2. Clean the data
   - Round float-type columns to two decimal places.
   - remove duplicate samples
   - Save the cleaned data into parquet file
3. Insert the data into SQL
   - Create postgresql database
   - Insert the data into the database

Submission requirement: 1. Jupyter Notebook 2. Parquet File 3. SQL file (optional)

```
# Required Package:
# psycopg2 2.9.10 (A PostgreSQL database adapter)
# pandas 2.0.3 (For data manipulation and analysis)
# sqlalchemy 2.0.37 (A SQL toolkit and Object Relational Mapper)
# pyarrow 14.0.1 (Provides support for efficient in-memory columnar data structures, part from Apache Arrow Objective)
import pandas as pd

#required for reading .xml files
import xml.etree.ElementTree as ET

#required for navigating machine's directory
import glob
import os.path

#required for communicating with SQL database
from sqlalchemy import create_engine
```

E: Extracting data from multiple sources

```
import boto3
import os

my_aws_access_key_id='ASIAYAAO5HRMPSPMX5OZ'
my_aws_secret_access_key='3BgvvyoDD+PEwIUmZwyfnut/CRwkCUa/t5BV8huP'
my_aws_session_token='IQoJb3JpZ2luX2VjEBoaCXVzLWVhc3QtMiJHMEUCIQCgvr7/uyXz5vonWNYq+l3RfXCaA/f/mcvokOVV6nquaQIgQaR3OV2UFys5s4XsdDWLKc7D/xYuM8
```

```
BUCKET_NAME = 'de300spring2025'   # Replace with your bucket name
S3_FOLDER = 'dinglin_xia/lab4_data/'             # The folder path in S3
LOCAL_DIR = './local-data/'      # Local directory to save files
```

```
def download_s3_folder(bucket_name, s3_folder, local_dir):
    """Download a folder from S3."""
    if not os.path.exists(local_dir):
        os.makedirs(local_dir)

    # List objects within the specified folder
    s3_resource = boto3.resource('s3',
                                 aws_access_key_id=my_aws_access_key_id,
                                 aws_secret_access_key=my_aws_secret_access_key,
                                 aws_session_token=my_aws_session_token)
    bucket = s3_resource.Bucket(bucket_name)

    for obj in bucket.objects.filter(Prefix=s3_folder):
        # Define local file path
        local_file_path = os.path.join(local_dir, obj.key[len(s3_folder):])

        if obj.key.endswith('/'):  # Skip folders
            continue

        # Create local directory if needed
        local_file_dir = os.path.dirname(local_file_path)
        if not os.path.exists(local_file_dir):
            os.makedirs(local_file_dir)

        # Download the file
        bucket.download_file(obj.key, local_file_path)
        print(f"Downloaded {obj.key} to {local_file_path}")
```

```
download_s3_folder(BUCKET_NAME, S3_FOLDER, LOCAL_DIR)
```

```
Downloaded dinglin_xia/lab4_data/used_car_prices1.csv to ./local-data/used_car_prices1.csv
Downloaded dinglin_xia/lab4_data/used_car_prices1.json to ./local-data/used_car_prices1.json
Downloaded dinglin_xia/lab4_data/used_car_prices1.xml to ./local-data/used_car_prices1.xml
Downloaded dinglin_xia/lab4_data/used_car_prices2.csv to ./local-data/used_car_prices2.csv
Downloaded dinglin_xia/lab4_data/used_car_prices2.json to ./local-data/used_car_prices2.json
Downloaded dinglin_xia/lab4_data/used_car_prices2.xml to ./local-data/used_car_prices2.xml
Downloaded dinglin_xia/lab4_data/used_car_prices3.csv to ./local-data/used_car_prices3.csv
Downloaded dinglin_xia/lab4_data/used_car_prices3.json to ./local-data/used_car_prices3.json
Downloaded dinglin_xia/lab4_data/used_car_prices3.xml to ./local-data/used_car_prices3.xml
```

Extract data from ./data/ folder

```
all_files = glob.glob('./data/*')

# Output the list of files
for file in all_files:
    print(file)
```

Function to extract data from one .csv file

```
def extract_from_csv(file_to_process: str) -> pd.DataFrame:

    return pd.read_csv(file_to_process)
```

Function to extract data from one .json file

```
def extract_from_json(file_to_process: str) -> pd.DataFrame:

    return pd.read_json(file_to_process, lines=True)
```

Function to extract data from one .xml file

```
def extract_from_xml(file_to_process: str) -> pd.DataFrame:
    dataframe = pd.DataFrame(columns = columns)
    tree = ET.parse(file_to_process)
    root = tree.getroot()
    for person in root:
        car_model = person.find("car_model").text
        year_of_manufacture = int(person.find("year_of_manufacture").text)
        price = float(person.find("price").text)
        fuel = person.find("fuel").text
        sample = pd.DataFrame({"car_model":car_model, "year_of_manufacture":year_of_manufacture, "price":price, "fuel":fuel}, index = [0])
        dataframe = pd.concat([dataframe, sample], ignore_index=True)
    return dataframe
```

Function to extract data from the ./data/ folder

```
def extract() -> pd.DataFrame:
    extracted_data = pd.DataFrame(columns = columns)
    #for csv files
    for csv_file in glob.glob(os.path.join(folder, "*.csv")):
        extracted_data = pd.concat([extracted_data, extract_from_csv(csv_file)], ignore_index=True)


    # JSON files
    for json_file in glob.glob(os.path.join(folder, "*.json")):
        extracted_data = pd.concat(
            [extracted_data, extract_from_json(json_file)], ignore_index=True
        )

    # XML files
    for xml_file in glob.glob(os.path.join(folder, "*.xml")):
        extracted_data = pd.concat(
            [extracted_data, extract_from_xml(xml_file)], ignore_index=True
        )

    return extracted_data
```

Extract the data

```
columns = ['car_model','year_of_manufacture','price', 'fuel']
folder = "data"
#table_name = "car_data"

# run
def main():
    data = extract()
    #insert_to_table(data, "car_data")

    return data

data = main()
```

```
data.head()
```

	car_model	year_of_manufacture	price	fuel

T: Transformation data and save organized data to .parquet file

```
staging_file = "cars.parquet"
staging_data_dir = "staging_data"
```

```
def transform(df):
    print(f"Shape of data {df.shape}")

    # truncate price with 2 decimal place (add your code below)

    # remove samples with same car_model (add your code below)

    print(f"Shape of data {df.shape}")

    # Ensure the staging directory exists before writing the Parquet file
    if not os.path.exists(staging_data_dir):
        os.makedirs(staging_data_dir)
        print(f"Directory '{staging_data_dir}' created.")

    # write to parquet
    df.to_parquet(os.path.join(staging_data_dir, staging_file))
    return df
```

```
# print the head of your data
df = transform(data)
df.head()
```

```
Shape of data (0, 4)
Shape of data (0, 4)
Directory 'staging_data' created.
```

	car_model	year_of_manufacture	price	fuel

L: Loading data for further modeling

Set Up PostgreSQL Locally

Step 1: Install PostgreSQL

- Windows: Download from MySQL Official Site {https://www.postgresql.org/download/}
- Mac:

```
brew install postgresql
brew services start postgresql
```

Then access PostgreSQL CLI

```
psql -U postgres
```

Note: if you don't have default "postgres" user, then create it manually by

```
default "postgres" user
```

or

```
sudo -u $(whoami) createuser postgres -s
```

Then create a database

```
CREATE DATABASE my_local_db;
\l  -- List all databases
```

Step 2: Create a User and Grant Privileges

In PostgreSQL CLI:

```
CREATE USER myuser WITH ENCRYPTED PASSWORD 'mypassword';
GRANT ALL PRIVILEGES ON DATABASE my_local_db TO myuser;
```

Step 3: Install Required Python Libraries

```
pip install pandas sqlalchemy pymysql psycopg2 mysql-connector-python
```

Utility function for writing data into the SQL database

```
# Database credentials
db_host = "localhost"
db_user = "your_user_name"
db_password = "your_password"
db_name = "your_database_name"

conn_string = f"postgresql+psycopg2://{db_user}:{db_password}@{db_host}/{db_name}"

engine = create_engine(conn_string)
```

```
# Test connection
df = pd.read_sql("SELECT * FROM pg_catalog.pg_tables;", con=engine)
print(df)
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/sqlalchemy/engine/base.py in __init__(self, engine, connection, _has_events, _allow_revalidate, _allow_autobegin)
    144             try:
--> 145                 self._dbapi_connection = engine.raw_connection()
    146             except dialect.loaded_dbapi.Error as err:

 37 frames
OperationalError: connection to server at "localhost" (127.0.0.1), port 5432 failed: Connection refused
        Is the server running on that host and accepting TCP/IP connections?
connection to server at "localhost" (::1), port 5432 failed: Cannot assign requested address
        Is the server running on that host and accepting TCP/IP connections?


The above exception was the direct cause of the following exception:

OperationalError                          Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/psycopg2/__init__.py in connect(dsn, connection_factory, cursor_factory, **kwargs)
    120
    121     dsn = _ext.make_dsn(dsn, **kwargs)
--> 122     conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
    123     if cursor_factory is not None:
    124         conn.cursor_factory = cursor_factory

OperationalError: (psycopg2.OperationalError) connection to server at "localhost" (127.0.0.1), port 5432 failed: Connection refused
        Is the server running on that host and accepting TCP/IP connections?
connection to server at "localhost" (::1), port 5432 failed: Cannot assign requested address
        Is the server running on that host and accepting TCP/IP connections?

(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

Next steps: Explain error

```
def insert_to_table(data: pd.DataFrame, conn_string:str, table_name:str):
    db = create_engine(conn_string) # creates a connection to the database using SQLAlchemy
    conn = db.connect() # Establishes a database connection
    data.to_sql(table_name, conn, if_exists="replace", index=False)
    conn.close()
```

```
# read from the .parquet file
```