MoP — Build Your First Automated ETL (CSV -> Clean CSV)

GOAL:

What you will automate

1. Extract: Read a raw CSV (e.g., telecom_raw.csv)
2. Transform: Clean & standardize (fill missing, parse dates, normalize text)
3. Load: Save a cleaned file (e.g., telecom_cleaned.csv)
4. Schedule: Run the job automatically at a time/interval

1) One-time setup

Create new folder: “training_auto_etl”
Create new notebook in jupyter browse in above folder: “auto_etl”

1.1 Install Required Packages

Run in Anaconda Prompt or Terminal:
pip install pandas schedule

1.2 Starter raw data (create telecom_raw.csv)

```
import pandas as pd

# Create the dataframe with your telecom data
data = {
    "customer_id": [1001, 1002, 1003, 1004, 1005],
    "data_used_gb": [5.2, None, 7.8, 15.6, 3.4],
    "calls_made": [25, 40, 32, 55, 18],
    "revenue_inr": [180, 280, 210, None, 120],
    "region": ["delhi", "Mumbai", "chennai", "DELHI", "Kolkata"],
    "date": ["2025/09/25", "2025-09-25", "25-09-2025", "2025-09-25", "2025-09-25"]
}

# Create a DataFrame
telecom_df = pd.DataFrame(data)

# Save to CSV
telecom_df.to_csv("telecom_raw.csv", index=False)

print("✅ telecom_raw.csv saved successfully.")
```

2) Write the automated ETL script (auto_etl.py)

ROLE: This Python script automates the cleaning and updating of a telecom dataset.
It reads a raw CSV file (telecom usage data), cleans it, and saves a cleaned version every few seconds automatically.

This process is called ETL:
Extract -> Transform -> Load

#Importing libraries

```
import os
import time
import pandas as pd
import schedule
from datetime import datetime
```

#Setting up file paths

```
RAW_PATH = "telecom_raw.csv"
OUT_DIR  = "output"
OUT_PATH = os.path.join(OUT_DIR, "telecom_cleaned.csv")
TMP_PATH = os.path.join(OUT_DIR, "telecom_cleaned.tmp.csv")
LOG_PATH = os.path.join(OUT_DIR, "etl_run.log")
```

#creates output folder

```
os.makedirs(OUT_DIR, exist_ok=True)
```

#Logs function

```
def log(msg: str):
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(LOG_PATH, "a", encoding="utf-8") as f:
        f.write(f"[{ts}] {msg}\n")
    print(f"[{ts}] {msg}")
```

#Data cleaning function

```
def clean_frame(df: pd.DataFrame) -> pd.DataFrame:
```

 # 1) Standardize text columns - Fix text formatting

```
    if "region" in df.columns:
        df["region"] = df["region"].astype(str).str.strip().str.title()
```

```
    # 2) Fill missing numeric values with robust medians – missing numbers

    for col in ["data_used_gb", "calls_made", "revenue_inr"]:
        if col in df.columns:
            if pd.api.types.is_numeric_dtype(df[col]) is False:
                df[col] = pd.to_numeric(df[col], errors="coerce")
            df[col] = df[col].fillna(df[col].median())


    # 3) Parse date; coerce + fill default - Fix date problems

    if "date" in df.columns:
        df["date"] = pd.to_datetime(df["date"], errors="coerce")
        df["date"] = df["date"].fillna(pd.Timestamp("2025-09-25"))

    # 4) De-duplicate but keep first; log how many were removed - Remove duplicates

    if {"customer_id", "date"}.issubset(df.columns):
        before = len(df)
        df = df.drop_duplicates(subset=["customer_id", "date"], keep="first")
        log(f"Deduplicated: removed {before - len(df)} duplicate row(s).")

    # 5) Clip clearly invalid ranges (safety net) - Keep data realistic

    if "data_used_gb" in df.columns:
        df["data_used_gb"] = df["data_used_gb"].clip(lower=0, upper=100)
    if "revenue_inr" in df.columns:
        df["revenue_inr"] = df["revenue_inr"].clip(lower=0)

    return df

# ETL JOB FUNCTION
def etl_job():
    try:
        log("Starting ETL...")
        if not os.path.exists(RAW_PATH):
            log(f"Raw file not found: {RAW_PATH}")
            return

        # EXTRACT
        df = pd.read_csv(RAW_PATH)

        # TRANSFORM
        df = clean_frame(df)

        # LOAD (atomic write: write tmp then rename -> avoids half-written files)
        df.to_csv(TMP_PATH, index=False)
```

```
        os.replace(TMP_PATH, OUT_PATH)

        log(f"ETL completed successfully. Rows written: {len(df)}.")
    except Exception as e:
        log(f"ETL failed: {e}")
```

 # SCHEDULING

```
    # --- DEMO MODE (runs every 10 seconds so students see it working) ---
    # schedule.every(10).seconds.do(etl_job)

    # --- PRODUCTION MODE (uncomment one of these) ---
    # schedule.every().day.at("09:00").do(etl_job)          # every day at 23 PM
    # schedule.every().hour.at(":00").do(etl_job)           # top of every hour

    # For continuity, keep every 10 seconds:
    #schedule.every(10).seconds.do(etl_job)

    #log("Scheduler started.")
    #while True:
        #schedule.run_pending()
        #time.sleep(1)

# For classroom demo, keep every 10 seconds for three times:
import time, schedule

schedule.clear()
schedule.every(10).seconds.do(etl_job)

runs = 3  # run 3 times then stop
for _ in range(runs):
    schedule.run_pending()
    time.sleep(10)   # wait for the next tick

print("✅ Done. Scheduler exited after", runs, "runs.")
```

3) Run the automation

From the folder telecom_auto_etl/:

You'll see log messages in the console and in output/etl_run.log.
A cleaned file appears at output/telecom_cleaned.csv.
Update telecom_raw.csv while the script runs—watch the cleaned file refresh automatically.

4) Verify the results

Open output/telecom_cleaned.csv — you should see:

- region normalized (e.g., delhi, DELHI → Delhi)
- Missing data_used_gb / revenue_inr filled with median
- Dates standardized to YYYY-MM-DD (bad formats fixed)
- Duplicates on (customer_id, date) removed (logged)
- No rows silently discarded