Data Pipelines & Workflow Automation

Introduction to Data Engineering

- **Definition:** Data Engineering is the practice of designing, building, and maintaining systems that move, store, and process data efficiently.
- **Goal:** Make **raw data** → **usable data** for analytics, BI, and machine learning.
- **Core Responsibilities:**
    1. **Data Ingestion** – bringing data in from multiple sources (APIs, databases, logs).
    2. **Data Transformation** – cleaning, aggregating, enriching.
    3. **Data Storage** – storing in a warehouse, lake, or DB.
    4. **Workflow Orchestration** – ensuring pipelines run on time and reliably.

Why important - Most ML/analytics projects fail without high-quality, automated data pipelines.

What is a Data Pipeline?

- A **data pipeline** is a series of steps that automate the flow of data from source → destination.
- Three core stages:
    1. **Extract (E):** Get raw data from sources (files, APIs, streaming).
    2. **Transform (T):** Clean, normalize, validate, and enrich data.
    3. **Load (L):** Store processed data into a database, data warehouse, or analytics system.

This is called the **ETL process**.

Sometimes the order changes to **ELT** (Extract → Load → Transform), common in modern cloud warehouses.

Example

```
import pandas as pd

# 1. Extract
df = pd.read_csv("raw_sales.csv")

# 2. Transform
df["Revenue"] = df["Quantity"] * df["Price"]   # create new column
df = df.dropna()                               # remove missing values

# 3. Load
df.to_csv("processed_sales.csv", index=False)
print("Pipeline finished ")
```

Workflow Automation

- Manual pipelines are error-prone and slow.
- Automation ensures:
    - **Scheduling** (e.g., daily/hourly runs).
    - **Error handling & retries**.
    - **Scalability** (parallel tasks).
    - **Monitoring** (logs, dashboards).

```
!pip install schedule

import schedule, time

import pandas as pd


def pipeline():

  data = pd.read_excel("/content/irisexcel1.xlsx")

  data = data.drop_duplicates()

  data = data.fillna(method="ffill")

  print("Rows:",len(data))
```

```
  print("Mean Values:\n",data.mean(numeric_only= True))


schedule.every(0.10).minutes.do(pipeline)


while True:
  schedule.run_pending()
  time.sleep(1), explain the code
```

“!pip install schedule

import schedule, time

import pandas as pd”

- `!pip install schedule` → Installs the **schedule** library (used for lightweight job scheduling in Python).
- `import schedule, time` → Imports required libraries.
    - `schedule` → to schedule tasks at specific intervals.
    - `time` → to control execution (sleep, waiting).
- `import pandas as pd` → for working with data.

“def pipeline():

  data = pd.read_excel("/content/irisexcel1.xlsx") # 1. Load Excel file

  data = data.drop_duplicates() # 2. Remove duplicate rows

  data = data.fillna(method="ffill") # 3. Fill missing values using forward-fill

  print("Rows:",len(data)) # 4. Print number of rows after cleaning

  print("Mean Values:\n",data.mean(numeric_only=True)) “# 5. Print mean of numeric columns

1. **Extract**: Reads data from an Excel file (`irisexcel1.xlsx`).
2. **Transform**:
    - Removes duplicate rows.
    - Fills missing values using **forward-fill** (`ffill`) → replaces missing value with the last known value.
3. **Load/Analysis**: Prints number of rows and mean values of all numeric columns.

This `pipeline()` function is basically a **mini ETL pipeline**.

“schedule.every(0.10).minutes.do(pipeline)”

- Schedules the `pipeline()` function to run **every 0.10 minutes (≈ 6 seconds)**.
- `schedule.every(X).minutes.do(func)` → tells scheduler to run a task repeatedly at given intervals.

“while True:

schedule.run_pending() # check if a job is due to run

time.sleep(1) “ # pause execution for 1 second before checking again

- This infinite loop keeps the program running.
- `schedule.run_pending()` → executes any jobs that are scheduled to run.
- `time.sleep(1)` → avoids high CPU usage by pausing for 1 second before checking again.