# Exercise: Build a Data Processing & Deployment Pipeline

**STEP – 1 :**

**raw_data.py**

```
import csv
def fetch():
data = [
{"Order ID": 101, "Customer": "arjun", "Order Date": "01-01-2024", "Ship Date": "05-01-2024", "Amount": "2500"},
{"Order ID": 102, "Customer": "priya", "Order Date": "02-01-2024", "Ship Date": "06-01-2024", "Amount": "1800"},
{"Order ID": 103, "Customer": "rahul", "Order Date": "05-01-2024", "Ship Date": "09-01-2024", "Amount": "3200"},
{"Order ID": 104, "Customer": "meena", "Order Date": "06-01-2024", "Ship Date": "10-01-2024", "Amount": "1500"},
{"Order ID": 105, "Customer": "suresh", "Order Date": "07-01-2024", "Ship Date": "11-01-2024", "Amount": "2750"},
{"Order ID": 106, "Customer": "anita", "Order Date": "08-01-2024", "Ship Date": "12-01-2024", "Amount": "3000"},
{"Order ID": 107, "Customer": "karthik", "Order Date": "09-01-2024", "Ship Date": "13-01-2024", "Amount": "2100"},
{"Order ID": 108, "Customer": "divya", "Order Date": "10-01-2024", "Ship Date": "14-01-2024", "Amount": "4000"},
{"Order ID": 109, "Customer": "manoj", "Order Date": "11-01-2024", "Ship Date": "15-01-2024", "Amount": "2200"},
{"Order ID": 110, "Customer": "rekha", "Order Date": "12-01-2024", "Ship Date": "16-01-2024", "Amount": "3500"},
   ]
 df = pd.DataFrame(data)
df.to_csv("raw_sales_data.csv", index=False)
if __name__ == "__main__":
   fetch()
```

**STEP – 2**

**transform_data.py**

```
import pandas as pd
def clean_data():
   df = pd.read_csv("raw_sales_data.csv")
   df = df.dropna()
```

```
df["Order Date"] = pd.to_datetime(df["Order Date"], errors="coerce", dayfirst=True)
.dt.strftime("%Y-%m-%d")
    df["Ship Date"] = pd.to_datetime(df["Ship Date"], errors="coerce", dayfirst=True)
.dt.strftime("%Y-%m-%d")
 df.columns = df.columns.str.lower().str.replace(" ", "_")
 df.to_csv("clean_sales_data.csv", index=False)
if __name__ == "__main__":
    clean_ data()
```

**YAML FILE**

```
trigger:
- main
pool:
  vmImage: ubuntu-latest

steps:
- task: Checkout@1

- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.10'
    addToPath: true

- script: |
    python -m pip install --upgrade pip
    pip install -r data_pipeline/requirements.txt
  displayName: 'Install dependencies'

- script: |
    cd data_pipeline
    python raw_data.py
  displayName: 'Fetch raw data'

- script: |
    cd data_pipeline
    python transform_data.py
  displayName: 'Transform and clean data'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: 'data_pipeline/clean_sales_data.csv'
    ArtifactName: 'CleanedSalesData'
    publishLocation: 'Container'
```

**Bonus Questions**

**1. Why is data cleaning important in real-time data processing?**

In real-time data processing, the value of data lies in its ability to support instant decision-making. However, raw data collected from sensors, applications, logs, social media, or IoT devices often contains noise, errors, duplicates, incomplete fields, and outliers. If this data is consumed without cleaning, it can cause incorrect metrics, wrong alerts, or even system failures in automation workflows. For instance, in a financial transaction monitoring system, uncleaned data might falsely flag a valid transaction as fraud, leading to customer dissatisfaction.

Data cleaning ensures that incoming data streams are accurate, consistent, and reliable before being processed further. It helps to remove duplicates, standardize formats (e.g., timestamps, currencies), correct invalid values, and fill in missing data. Clean data improves the quality of analytics, enhances machine learning model performance, and ensures that real-time dashboards reflect reality rather than noise. Moreover, in mission-critical domains like healthcare monitoring or fraud detection, clean data can directly impact safety and trust. Therefore, data cleaning is not just a preprocessing step but a critical requirement for maintaining data integrity in real-time processing pipelines.

**2. What are pipeline artifacts and how are they used in DevOps workflows?**

In DevOps workflows, a pipeline is composed of multiple stages such as build, test, and deploy, and each stage may generate outputs that need to be reused later. These outputs are called pipeline artifacts. An artifact could be a compiled binary, Docker image, configuration file, test report, or even processed datasets. They are stored and passed between pipeline stages to maintain consistency, traceability, and repeatability of deployments.

For example, when a developer pushes code, the pipeline builds the application and generates a build artifact (e.g., .jar, .exe, or Docker image). Instead of rebuilding the code again during deployment, the same artifact is reused in the deployment stage, ensuring that the exact code version that passed testing is deployed in production. This reduces risks, avoids mismatches, and guarantees that “what was tested is what gets deployed.”

Pipeline artifacts are also useful for auditing and rollback purposes. Since artifacts can be versioned and stored in repositories (e.g., Azure Artifacts, JFrog, Nexus, or container registries), teams can trace back to a specific version if an issue occurs. Additionally, they allow sharing between pipelines, meaning one team's output (e.g., a shared library or ML model) can serve as another team's input. In short, artifacts form the bridge between different stages of the DevOps lifecycle, ensuring smooth and reliable software delivery.

**3. How would you modify the pipeline to store the cleaned data into Azure Blob Storage?**

To store cleaned data into Azure Blob Storage as part of the pipeline, we need to extend the CI/CD pipeline definition with a new task that uploads the processed data into a Blob container. After the data cleaning step finishes, a storage task can be triggered to package and transfer the output (CSV, JSON, or Parquet files) to Blob Storage.

This can be achieved using Azure CLI tasks or AzureFileCopy@4 task in Azure DevOps. First, the cleaned dataset must be stored as a pipeline artifact or intermediate file. Then, using secure credentials (storage account key, SAS token, or service principal), the pipeline uploads the file to the designated container within Azure Blob Storage. Storing credentials in Azure DevOps secure variables or Key Vault integration ensures security.