Bonus Questions – Azure DevOps & Data Processing

1. Why is Data Cleaning Important in Real-Time Data Processing?

Definition

Data cleaning (or data preprocessing) is the process of **identifying, correcting, or removing inaccurate, incomplete, or irrelevant data** before analysis or further processing.

In **real-time data processing**, where data flows continuously from various sources like IoT devices, e-commerce platforms, or financial systems, **clean and accurate data** is critical for making correct business decisions

Importance

Aspect	Without Cleaning	With Cleaning
Accuracy	Wrong or duplicate data leads to incorrect results.	Ensures reliable insights.
Consistency	Different formats and missing fields cause confusion.	Standardizes and organizes data.
Performance	Raw, unoptimized data slows down pipelines.	Speeds up processing.
Decision-Making	Poor-quality data leads to bad business decisions.	Enables accurate decision-making.

Example

Suppose an e-commerce dataset like this:

Transaction ID	Date	Customer Name	Product	Quantity	Price	Total Amount
T005	2024-01-15	Aryan	Tablet	**1**	**20,000**	**20,000**
T006	2024-01-18	Kiran	Smartphone	**-1**	26,000	-26,000

- Without cleaning, the **negative quantity** (-1) would lead to **wrong revenue calculations**.

- During cleaning, we fix invalid values, handle duplicates, fill missing fields, and correct data types.

Key Steps in Real-Time Data Cleaning

1. **Remove Duplicates** → Ensures unique transactions.
2. **Handle Missing Values** → Fill or drop nulls.
3. **Correct Data Types** → Convert strings, numbers, and dates to the right formats.
4. **Standardize Formats** → Consistent naming, date, and currency formats.
5. **Validate Business Rules** → Example: Quantity ≥ 1, Price ≥ 0.

Real-Time Use Cases

- **Banking & Payments:** Prevents fraudulent transactions caused by duplicate or wrong data.
- **E-commerce:** Ensures correct order details and payment amounts.
- **Healthcare:** Avoids medical errors caused by incomplete patient data.

2. What Are Pipeline Artifacts and How Are They Used in DevOps Workflows?

Definition

In **Azure DevOps**, **artifacts** are **files or outputs** created during the build or release process that you want to **save, share, or deploy later**.

For example, when your **data_processing.py** script runs in the pipeline, it creates:

- raw_sales_data.csv
- cleaned_sales_data.csv

These files are **pipeline artifacts**.

Why Artifacts Are Important

- They **store processed outputs** for later use.
- They allow **team collaboration** since artifacts can be **downloaded** by other teams.
- They support **multi-stage pipelines**, where one stage produces data and another consumes it.

How Artifacts Work in Azure Pipelines

1. **Build Stage**
    - Your pipeline runs data_processing.py.
    - The script generates CSV files.
2. **Publish Artifacts**
    - Use the PublishPipelineArtifact task in azure-pipelines.yml.
    - Saves files for later use.
3. **Download Artifacts**
    - Later pipeline stages (e.g., testing, reporting) **consume** the saved artifacts.

Example

```
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: "$(System.DefaultWorkingDirectory)/data"
    artifact: "sales-data"
```

Use Cases

- Store processed data files.
- Share trained machine learning models.
- Save test reports and deployment packages.

3. How Would You Modify the Pipeline to Store the Cleaned Data into Azure Blob Storage?

To **automatically upload your cleaned data** to **Azure Blob Storage** after your pipeline runs, you need to:

Step 1 — Create a Storage Account

1. Go to Azure Portal.
2. Search **"Storage Accounts"** → Click **Create**.

3. Choose **Resource Group**, **Name**, and **Region**.
4. Enable **Blob Storage** service.

Step 2 — Get the Connection String

- Navigate to **Storage Account → Access Keys**.
- Copy the **Connection String** or **SAS Token**.

Step 3 — Update azure-pipelines.yml

Add a step after your script execution to **upload cleaned data**:

```
steps:
  - task: AzureCLI@2
    inputs:
      azureSubscription: "<Your-Service-Connection-Name>"
      scriptType: bash
      scriptLocation: inlineScript
      inlineScript: |
        az storage blob upload \
          --account-name <your-storage-account> \
          --container-name sales-data \
          --file cleaned_sales_data.csv \
          --name cleaned_sales_data.csv \
          --auth-mode key \
          --account-key $(AZURE_STORAGE_KEY)
```

Step 4 — Securely Store the Storage Key

- Go to **Azure DevOps → Project Settings → Pipelines → Library → Variables**.
- Add AZURE_STORAGE_KEY as a **secret variable**.
- Reference it in your YAML file like above.

Step 5 — Verify Upload

- Go to **Azure Portal → Storage Account → Containers**.
- Check if cleaned_sales_data.csv appears in the container.

Benefits

Benefit	**Why It Matters**
Automation	No need to manually upload data.
Centralized Storage	All cleaned data stored in one place.
Integration	Easily connect with Power BI, Databricks, or ML pipelines.
Scalability	Handles millions of CSV records seamlessly.

Final Takeaways

- Data cleaning is **essential** for accurate real-time analytics.
- Pipeline artifacts **store and share processed outputs** across pipeline stages.
- Azure Blob Storage **integrates seamlessly** with pipelines to automate cleaned data uploads.