#### **Problem 1: Parse Fixed-Width File**

The goal of this problem is to process a **fixed-width text file** (where each field has a specific number of characters) based on a specification file (spec.json) and generate a **CSV file** with delimited data. Here’s the step-by-step explanation:

#### **1. What is a Fixed-Width File?**

A fixed-width file is a text file where:

* Each column has a pre-defined fixed width (number of characters).
* Padding (usually spaces) is used to ensure each column has consistent width.  
    
    
  John Doe 32NY1234567890 123 Main St. 07-04-1990 1.23E+10 ACTIVE

The above line has fields like:

* Name: John Doe
* Age: 32
* State: NY
* Identifier: 1234567890 ... all with fixed lengths.

**2. Objective**

* Read the spec.json file:
  + **ColumnNames**: Names of the fields in the file (e.g., f1, f2, etc.).
  + **Offsets**: The widths of each field (e.g., 5, 12, etc.).
  + **FixedWidthEncoding**: The encoding of the input file.
  + **DelimitedEncoding**: The encoding of the output CSV.
  + **IncludeHeader**: Whether to include headers in the output CSV.
* Parse the fixed-width file (data/input\_fixed\_width.txt) into structured data based on the offsets.
* Generate a CSV file (output.csv) with the data in delimited format (e.g., commas).

**3. Challenges**

1. **Encoding Issues**: Ensure the fixed-width file is correctly read with its specified encoding (windows-1252 in this case).
2. **Parsing Logic**: Split each line into fields based on the offsets without using libraries like pandas.
3. **Formatting**: Output the delimited data in the correct encoding (utf-8) and include headers if specified.  
     
     
     
   **4. Steps in the Solution**
4. **Read the spec.json file**: Extract column names and offsets.
5. **Process the input file**: For each line in the fixed-width file:
   * Use the offsets to slice the string into fields.
6. **Write the CSV file**:
   * Use Python's built-in csv module to write the parsed data into a CSV file.
7. **Dockerize the Solution**: Provide a Dockerfile to containerize the script for easy deployment.

**Problem 2: Data Processing**

The goal of this problem is to **anonymize sensitive data** in a CSV file, handle large files (up to 2GB or more), and demonstrate scalability for massive datasets using distributed computing tools.  
  
  
**1. What Does It Mean to Anonymize Data?**

Anonymization means replacing sensitive fields (e.g., first\_name, last\_name, address) with non-identifiable values like hashed strings, randomized data, or pseudonyms.

**Example:**

* Original  
    
  John,Doe,123 Main St.,1990-07-04

Anonymized:  
  
XXXX,XXXX,XXXX XXXX,1990-07-04

#### **2. Objective**

* Generate a sample CSV file containing columns:
  + first\_name, last\_name, address, date\_of\_birth.
* Write a script to:
  + Read the CSV file.
  + Anonymize first\_name, last\_name, and address.
  + Save the anonymized data into a new CSV file.
* Handle **large datasets** (e.g., 2GB+ files):
  + Process the file line-by-line or in chunks to avoid memory issues.
  + Demonstrate scalability using distributed tools like **Apache Spark** or **Dask**.

**3. Challenges**

1. **Handling Large Files**: A 2GB file cannot be fully loaded into memory on most machines. The script must process data incrementally.
2. **Distributed Computing**:
   * For massive datasets (e.g., 100GB+), use a framework like Spark to distribute the work across multiple nodes.
3. **Anonymization**:
   * Replace sensitive fields while preserving the structure of the dataset.

#### **4. Steps in the Solution**

1. **Generate the Input File**: Create a sample CSV file with the specified columns.
2. **Write the Anonymization Script**:
   * Use Python’s standard library or libraries like csv for small files.
   * Use distributed tools (e.g., PySpark or Dask) for scalability.
3. **Demonstrate Scalability**:
   * Run the script on a small file locally.
   * Simulate larger datasets to showcase scalability.

#### **Scaling Problem 2 Using Distributed Tools**

##### **Why Use Spark?**

Spark is a distributed computing platform that can:

* Read large files in chunks distributed across nodes.
* Perform transformations (e.g., anonymization) efficiently.

##### **Steps to Implement Spark Solution**

1. Write a **PySpark** script to read the CSV file.
2. Apply a transformation function to anonymize sensitive columns.
3. Save the anonymized data back to a CSV.