Backend Intern Assignment - CapIA.ai

Welcome to the CapIA.ai backend intern assignment. This task is designed to assess your understanding of backend development principles and your ability to apply them in a practical scenario.

Objective

The objective of this assignment is to test your skills in data normalization and transformation using Python

Problem Statement:

```python
Create a Python program that can read CSV files with different delimiters and normalize the data, while correctly handling cases where delimiters appear within field values (especially in numbers and text).
```

Target Schema:

```
transaction_date: datetime    # Target format: YYYY-MM-DD
description: str             # Any text
amount: decimal             # Numeric value (e.g., 23500.00)
currency: str               # 3-letter currency code
status: str                 # lowercase status
```

Test Files:

1. Comma-delimited (test1.csv):

```
Transaction_Date,Description,Amount,Currency,Status
2024-01-15,Office Supplies,"1,234.56",USD,COMPLETED
2024-01-16,Software License,"2,500.00",USD,pending
2024-01-17,"Lunch, Meeting","1,750.50",USD,COMPLETED
```

2. Semicolon-delimited (test2.csv):

```
Transaction_Date;Description;Amount;Currency;Status
2024-01-15;Office Supplies;1.234,56;EUR;COMPLETED
2024-01-16;Software License;2.500,00;EUR;PENDING
2024-01-17;Lunch Meeting;1.750,50;EUR;completed
```

3. Pipe-delimited (test3.csv):

```
Transaction_Date|Description|Amount|Currency|Status
2024-01-15|Office Supplies|1,234.56|USD|COMPLETED
2024-01-16|Software, License|2,500.00|USD|PENDING
2024-01-17|Lunch Meeting|1,750.50|USD|completed
```

Requirements:

1. Delimiter Handling:

```
    """
    - Auto-detect the delimiter (comma, semicolon, pipe or any
other delimiter). Do not hardcode the delimiter type
    - Correctly handle delimiters within quoted fields
    - Preserve numeric values with thousand separators
    Example:
        "1,234.56" should become 1234.56, not split into ["1",
"234.56"]
        "Software, License" should remain one field, not split
    """
```

2. Data Normalization:

```
    """
    - Convert column names to snake_case ( handle special
character as well)
        "Transaction_Date" -> "transaction_date"
    - Standardize date format to YYYY-MM-DD
    - Convert amount strings to decimal numbers
        "$1,234.56" -> 1234.56
        "1.234,56" -> 1234.56
    - Standardize status to lowercase
        "COMPLETED" -> "completed"
    """
```

Example Expected Output:

```
# Input row: "2024-01-15,Office Supplies,"1,234.56",USD,COMPLETED"
# Should become:
{
    'transaction_date': datetime(2024, 1, 15),
    'description': 'Office Supplies',
    'amount': Decimal('1234.56'),
    'currency': 'USD',
    'status': 'completed'
}
```

Bonus Challenge: Handling Files Without Headers

Additional test file (no_header.csv):

```
2024-01-15,Office Supplies,"1,234.56",USD,COMPLETED
2024-01-16,Software License,"2,500.00",USD,PENDING
```

Requirements:

```
    """
    - Detect if file has headers
    - Map columns based on position or content pattern
      Note that the order of column could be different
    - Apply same normalization rules as above

    column_mapping example:
    {
        0: 'transaction_date',
        1: 'description',
        2: 'amount',
        3: 'currency',
        4: 'status'
    }
    """
```

Evaluation Criteria:

1. Core Challenge (70% of score):
    - Correct delimiter detection
    - Proper handling of numbers with thousand separators
    - Accurate data normalization
    - Clean code organization

2. Bonus Challenge (30% of score):
    - Successful handling of headerless files
    - Accurate column mapping
    - Error handling for mismatched columns