Financial Data Parser

Project Overview

Build a robust financial data parsing system that can process Excel files, intelligently detect data types, handle various formats, and store data in optimized structures for fast retrieval.

Learning Objectives

- Excel file processing with multiple libraries
- Data type detection and validation
- Format parsing for financial data
- Data structure optimization for performance
- Error handling and data quality assurance

Project Requirements

Phase 1: Basic Excel Processing

Files to Process:

- `KH_Bank.XLSX`
- `Customer_Ledger_Entries_FULL.xlsx`

Tasks:

1. Read both Excel files using pandas and openpyxl
2. Handle multiple worksheets within each file
3. Display basic file information (sheets, dimensions, column names)

Phase 2: Data Type Detection

Implement intelligent column classification:

String Columns:

- Account names, descriptions, categories
- Transaction references, notes
- Company names, addresses

Number Columns:

- Financial amounts (revenue, expenses, assets)
- Quantities, percentages, ratios
- Account balances, transaction amounts

Date Columns:

- Transaction dates, reporting periods

- Due dates, maturity dates
- Fiscal year-end dates

Detection Strategy:

```
def detect_column_type(column_data):
    # Remove null values for analysis
    # Try parsing as dates first
    # Try parsing as numbers (handle currency symbols)
    # Default to string if neither works
    # Return confidence score for each type
```

Phase 3: Format Parsing Challenges

Amount Formats to Handle:

- `$1,234.56` (US currency)
- `€1.234,56` (European format)
- `₹1,23,456.78` (Indian format)
- `(1,234.56)` (Negative in parentheses)
- `1234.56-` (Trailing negative)
- `1.23K`, `2.5M`, `1.2B` (Abbreviated amounts)

Date Formats to Handle:

- `MM/DD/YYYY`, `DD/MM/YYYY`
- `YYYY-MM-DD`, `DD-MON-YYYY`
- `Quarter 1 2024`, `Q1-24`
- `Mar 2024`, `March 2024`
- Excel serial dates (44927 = Jan 1, 2023)

Phase 4: Data Structure Implementation

Requirements:

- Fast lookup by multiple criteria
- Memory efficient storage
- Support for range queries (date ranges, amount ranges)
- Easy aggregation capabilities

Suggested Structures:

1. **Pandas DataFrame with MultiIndex**
2. **Dictionary-based indexing**
3. **SQLite in-memory database**
4. **Custom hash tables for specific queries**

Implementation Guide

Step 1: Environment Setup

```
# Required libraries
import pandas as pd
import openpyxl
import numpy as np
import sqlite3
import re
from datetime import datetime
from decimal import Decimal
import locale
```

Step 2: File Reading Class

Create an `ExcelProcessor` class with methods:

- `load_files(file_paths)`
- `get_sheet_info()`
- `extract_data(sheet_name)`
- `preview_data(rows=5)`

Step 3: Type Detection Engine

Build a `DataTypeDetector` class:

- `analyze_column(data)`
- `detect_date_format(sample_values)`
- `detect_number_format(sample_values)`
- `classify_string_type(sample_values)`

Step 4: Format Parser

Implement `FormatParser` class:

- `parse_amount(value, detected_format)`
- `parse_date(value, detected_format)`
- `normalize_currency(value)`
- `handle_special_formats(value)`

Step 5: Storage System

Design `DataStorage` class:

- `store_data(dataframe, metadata)`
- `create_indexes(columns)`
- `query_by_criteria(filters)`
- `aggregate_data(group_by, measures)`

Sample Data Structure

```
# Example of optimized storage
class FinancialDataStore:
    def __init__(self):
        self.data = {}
        self.indexes = {}
        self.metadata = {}

    def add_dataset(self, name, df, column_types):
        # Store main data
        self.data[name] = df

        # Create indexes for fast lookup
        self.indexes[name] = {
            'date_index': {},
            'amount_index': {},
            'category_index': {}
        }

        # Store column metadata
        self.metadata[name] = column_types
```

Deliverables

Foundation

Excel file reading functionality

Basic data preview and inspection

Initial data type detection

Advanced Parsing

Robust amount parsing (multiple formats)

Date parsing with format detection

Error handling and validation

Storage Optimization

Implement multiple storage strategies

Performance benchmarking

Query optimization

Integration & Testing

Complete integration testing

Performance analysis report

Documentation and code review

Sample Test Cases

```
# Test amount parsing
test_amounts = [
    "$1,234.56",
    "(2,500.00)",
    "€1.234,56",
    "1.5M",
    "₹1,23,456"
]

# Test date parsing
test_dates = [
    "12/31/2023",
    "2023-12-31",
    "Q4 2023",
    "Dec-23",
    "44927"  # Excel serial
]
```

Resources and References

- **Pandas Documentation**: Data manipulation basics
- **OpenPyXL Guide**: Advanced Excel operations
- **Regular Expressions**: Pattern matching for formats
- **SQLite Tutorial**: Database operations
- **Performance Profiling**: cProfile and memory_profiler

Getting Started

1. Use project repository
2. Install required dependencies
3. Sample financial data files are inside data/sample folder
4. Start with Phase 1 implementation