Task Two: Data Consolidation Pipeline

Overview

Two implementations of the consolidation pipeline:

1. `consolidate_payments.py` - Basic batch processing from CSV files
2. `consolidate_payments_streaming.py` - Streaming version with different ingestion patterns

Both merge four data sources into a unified table for fraud detection.

Data Sources

Four sources with different timing patterns:

1. **Payment Transactions**: Real-time API (simulated as 15-min batches)
2. **Account Details**: Daily CSV batch (arrives at 6 AM)
3. **External Risk Feed**: Irregular batch updates
4. **Fraud Labels**: Delayed 30-60 days, mixed formats (API, CSV, Excel)

Consolidation Logic

- Join payments to source and destination account details
- Add risk flags for both accounts
- Include fraud labels where available
- Handle missing data (risk_flag=0, fraud_flag=False for missing)

Implementation

Basic Version (`consolidate_payments.py`)

Simple class-based design: - Separate methods for loading each source - Validation and error handling - LEFT joins from payments base table - Logging for debugging

Handles data quality: - Schema validation (checks required columns) - Type conversions (dates, amounts) - Deduplication (latest per account/payment) - Defaults for missing values

Streaming Version (`consolidate_payments_streaming.py`)

Simulates different ingestion patterns using generators and state management:

- `PaymentAPIStream`: Yields transaction chunks (simulates real-time API)
- `FraudLabelAPIStream`: Streams delayed fraud labels (45-day lag)
- `RiskFeedBatchLoader`: Handles irregular batch arrivals
- `StreamingPaymentConsolidator`: Maintains state buffers and consolidates

State management: - Payments: buffered until consolidation - Accounts: daily snapshot refresh - Risk flags: keep highest per account - Fraud labels: append-only, latest per payment

In production, the generator patterns could be replaced with Kafka consumers or CDC streams.

Usage

Install dependencies:

```
pip install -r requirements.txt
```

Run basic version:

```
python consolidate_payments.py
# Output: unified_table.csv
```

Run streaming version:

```
python consolidate_payments_streaming.py
# Output: unified_table_streaming.csv
```

Both produce the same output (15 records, 4 fraud cases), just different approaches.

Data Files

Input CSVs in root directory: - `payment_transactions.csv` (15 transactions) - `account_details.csv` (22 accounts) - `external_risk_feed.csv` (6 flagged accounts) - `historical_fraud_cases.csv` (4 fraud cases)

Output: - `unified_table.csv` or `unified_table_streaming.csv` (15 consolidated records)

Design Choices

LEFT joins: Never drop payment transactions, even if some account details are missing. Helps surface data quality issues.

Deduplication: Keep latest for accounts, highest risk for risk flags, latest for fraud labels. Conservative approach for fraud detection.

Defaults: risk_flag=0 (assume clean), fraud_flag=False (unlabeled fraud). Makes training data clearer.

Streaming state: Maintain buffers in-memory for this prototype. In production would use Kafka + Delta Lake or similar.

Production Scaling

Current: Single-process pandas (works for <1M rows)

To scale: - Replace pandas with Spark for distributed processing - Replace CSV with Parquet or Delta Lake - For streaming: Kafka → Spark Streaming → Delta Lake - Add monitoring (Prometheus metrics for throughput, quality, latency) - Deploy on Kubernetes for auto-scaling

Results

Both scripts produce identical output: - 15 consolidated records - 4 fraud cases identified (26.7% rate) - 100% account join success - All required columns present