Peak Power Services – YE Reconciliation & Risk Analytics Dashboard (FY 2025)

This repository contains a self-service year-end reconciliation framework built from the perspective of an Accounting Associate with data engineering expertise.

It simulates Peak Power Services' 2025 year-end close for a Florida-based, project-driven services company and demonstrates how to:

- Reconcile subledgers and schedules to the general ledger
- Systematically identify and prioritize discrepancies
- Support the Controller and Billing Supervisor with controller-ready summaries and suggested journal entries
- Lay a foundation for an ERP-style, warehouse-backed close process using CSVs and BigQuery-style modeling

The design moves typical spreadsheet workflows toward a more structured, reproducible, and scalable internal control environment.

1. Repository Structure

A typical layout of this project:

```
peak-power-ye-reconciliation/
│
├── dashboard.py                 # Streamlit YE Reconciliation
Dashboard
├── validator.py                 # Validation + risk scoring pipeline
(reads/writes CSV or BigQuery)
├── scoring_rules.py             # Centralized risk models and
probability logic (AP, Bank, Tax, Leases)
├── peak_data_gen.py             # Synthetic data generator for FY
2025 (optional if data already present)
├── requirements.txt             # Python dependencies
│
├── data/                        # Raw input data (pre-validation)
│   ├── AP_Subledger.csv
│   ├── Bank_Transactions.csv
│   ├── Tax_Detail.csv
│   ├── Lease_Schedule.csv
│   └── GL_Trial_Balance.csv
│
└── output/                      # Post-validation outputs with risk
scores
    ├── AP_with_risk.csv
    ├── Bank_with_risk.csv
    ├── Tax_with_risk.csv
    └── Lease_with_risk.csv
```

- `validator.py` reads from `data/` (or BigQuery tables), applies checks and scoring via `scoring_rules.py`, and writes enriched CSVs to `output/`.

- `dashboard.py` is presentation-only; it reads the `output/*.csv` files and surfaces the risk profile, exceptions, and suggested journal entries.

2. Business Context and Goals

Peak Power Services is a Florida-based services company with:

- High-volume accounts payable
- Multiple vendors and projects
- Sales and use tax exposure across Florida jurisdictions
- Several operating leases under ASC 842

The project simulates the Accounting Associate's role in year-end close:

- Tie subledgers (AP, leases) and bank activity to the GL
- Validate sales and use tax calculations and liability balances
- Highlight high-risk discrepancies instead of scanning every line manually
- Provide controller-ready journal entry suggestions and summarized risk views

The intent is to demonstrate how a mid-sized accounting function can combine GAAP, tax, and lease expertise with light data engineering to "work smarter, not harder" at year-end.

3. Data Model Details

All core data originates from five CSVs in the `data/` folder (or the equivalent tables in BigQuery). These are transformed and enriched into the `output/` files.

3.1 AP_Subledger.csv

Represents AP detail for FY 2025, including invoice amounts, tax, GL coding, and payment status.

Column	Description
Invoice_ID	Unique identifier for each invoice
Vendor	Vendor name
Invoice_Date	Date invoice was issued
Due_Date	Due date per vendor terms
Project_ID	Internal project or job identifier
GL_Account	Account to which the invoice is coded (expense, project, etc.)
Amount_Before_Tax	Net invoice amount before tax
Tax_Jurisdiction	Jurisdiction used for tax (e.g., Hillsborough, Pinellas, Orange, Other_FL)
Tax_Amount	Tax amount recorded on the invoice
Total_Invoice_Amount	Amount_Before_Tax + Tax_Amount
Paid_Flag	Boolean: whether invoice is marked as paid
Paid_Date	Date payment cleared (if paid)
Expected_Total	Recalculated expected total (validation check)
Total_Mismatch_Flag	Flag if Total_Invoice_Amount ≠ Expected_Total beyond tolerance
Unpaid_AsOfYE	Flag if unpaid as of year-end

AP_Match_Key	Synthetic match key for linking to Bank_Transactions
Bank_Match_Status	Status summarizing match result vs bank (e.g., Matched, Paid_No_Bank)

The validator will derive additional columns for flags and risk scoring.

3.2 Bank_Transactions.csv

Represents cash disbursements relevant to AP payments.

Column	Description
Vendor	Vendor name (should align with AP vendor)
Description	Free-text description for the payment
Amount	Payment amount (negative for cash outflows, if modeled that way)
Check_ACH	Payment method (Check, ACH, etc.)
Cleared_Flag	Boolean: whether the payment cleared the bank
Match_Key	Key used to match back to AP_Match_Key in AP_Subledger
Duplicate_Payment	Flag indicating suspected duplicate payment
Duplicate_Payment_Flag	Boolean flag for duplicate payment issues

The validator will detect missing invoices, duplicate payments, and amount mismatches against AP.

3.3 Tax_Detail.csv

Represents invoice-level tax detail and supports Florida sales & use tax validation.

Column	Description
Invoice_ID	Links back to AP_Subledger
State	For this project, this is used as Tax_Jurisdiction (Hillsborough, etc.)
Taxable_Amount	Amount subject to tax
Tax_Rate	Rate applied on the invoice
Calculated_Tax	Tax amount recorded/expected from subledger
GL_Tax_Liability_Account	GL account to which the tax is posted
Recalc_Tax	Recalculated tax using reference jurisdiction table
Tax_Mismatch_Flag	Flag if Calculated_Tax ≠ Recalc_Tax beyond tolerance
Correct_Tax_Rate	Jurisdiction-correct tax rate (from reference table)
Correct_Rate_Flag	TRUE if applied Tax_Rate equals Correct_Tax_Rate; FALSE otherwise

Florida tax reference used in the pipeline:

Tax_Jurisdiction	Surtax_2025	Total_Tax_Rate_2025
Hillsborough	0.015	0.075
Pinellas	0.010	0.070
Orange	0.005	0.065
Other_FL	0.000	0.060

3.4 Lease_Schedule.csv

Represents ASC 842 lease amortization schedules.

Column	Description
Lease_ID	Unique identifier per lease
Start_Date	Lease start date
Payment_Date	Payment date for each period
Lease_Payment	Total payment for the period
Interest_Portion	Interest portion of the payment
Principal_Portion	Principal portion of the payment
Ending_Lease_Liability	Closing lease liability after the payment
ROU_Asset_Balance	Closing ROU asset balance
IP_Sum	Calculated Interest + Principal (check column)
IP_Sum_Diff	Difference between IP_Sum and Lease_Payment
IP_Sum_Mismatch_Flag	TRUE if IP_Sum_Diff exceeds tolerance
Sequence_Check	Label to indicate if periods follow expected sequence (OK / Sequence Error)

The validator aggregates lease liability and ROU asset totals and compares them to GL.

3.5 GL_Trial_Balance.csv

Represents a summarized trial balance for key accounts at year-end.

Column	Description
Account	GL account name (e.g., Accounts Payable, Cash, etc.)

Ending_Balance	Year-end balance for that account

These balances are used as the control totals to reconcile AP, tax, and lease schedules.

4. Scoring and Risk Models (scoring_rules.py)

All scoring logic and probability mapping live in `scoring_rules.py`. This makes the risk models:

- Centralized
- Reusable
- Easy to adjust for different materiality or control environments

Each model (AP, Bank, Tax, Leases) operates on a set of flags and applies a weighted risk score:

- `risk_score = min(100, 5 + Σ(weight_i * flag_i))`
    - Where each flag is treated as 1 if true, 0 if false

Risk bands are shared across models:

Risk Score Range	Probability Band (Approx.)	Priority
0–20	5–15%	Low
21–40	20–40%	Medium
41–70	50–80%	High
71–100	85–99%	Critical (Top list first)

4.1 Model A – AP ↔ GL Discrepancy Risk

Flags per invoice (as columns in AP_with_risk.csv):

- `missing_in_GL` – AP invoice has no corresponding GL entry
- `amount_mismatch` – AP expected total vs GL posting differ beyond set tolerance
- `late_posting` – Posting date falls significantly after invoice date or outside the period

- `duplicate_invoice_number` – Same vendor and invoice pattern appears more than once
- `unusual_GL_account` – Invoice posted to an atypical GL account given the vendor/project

Example weight matrix for AP risk:

Condition	Weight (risk points)	Implied Probability of Discrepancy
No flags	0	5%
`late_posting` only	10	15%
`amount_mismatch` only	30	40%
`unusual_GL_account` only	25	35%
`duplicate_invoice_number` only	40	60%
`missing_in_GL` only	60	80%
`missing_in_GL` + any other flag	80	90–95%
`amount_mismatch` + `duplicate_invoice_number`	70	85–90%
3 or more flags (any combination)	90	95–99%

`scoring_rules.py` exposes a function that takes an AP row (Series) and returns:

- Risk score
- Probability band
- Dominant cause(s)

The `validator.py` script merges these results into `AP_with_risk.csv`.

4.2 Model B – Bank ↔ AP/Cash Discrepancy Risk

Flags per bank transaction:

- `no_matching_invoice` – Bank payment has no matching AP invoice (via Match_Key or vendor/amount logic)
- `invoice_marked_paid_but_no_bank_txn` – AP shows paid, but corresponding cash movement is missing
- `duplicate_payment` – Multiple payments appear for what should be a single AP obligation
- `amount_mismatch` – Paid amount differs from invoice amount beyond tolerance
- `unusual_vendor_payment` – Vendor has unusual payment frequency or amount pattern

Weight matrix:

Condition	**Weight (risk points)**	**Implied Probability**
No flags	0	5%
`unusual_vendor_payment` only	10	15%
`amount_mismatch` only	25	35%
`invoice_marked_paid_but_no_bank_txn` only	40	60%
`no_matching_invoice` only	50	70%
`duplicate_payment` only	60	80%
`duplicate_payment` + `amount_mismatch`	75	90%
`no_matching_invoice` + any other flag	80	90–95%
3 or more flags	90	95–99%

`Bank_with_risk.csv` then contains these computed scores and can be filtered in the dashboard.

4.3 Model C – Sales & Use Tax Discrepancy Risk

Flags per tax record (per invoice):

- `rate_mismatch` – Applied Tax_Rate ≠ Correct_Tax_Rate for the jurisdiction
- `tax_missing` – Taxable_Amount > 0 but Calculated_Tax = 0
- `tax_on_nontaxable_item` – Tax calculated where invoice line should not be taxable (simulated)
- `jurisdiction_missing` – Jurisdiction not found or invalid
- `gl_tax_diff_flag` – Jurisdiction/period-level sum of Calculated_Tax ≠ GL Tax Payable balance beyond tolerance

Weight matrix:

Condition	Weight (risk points)	Implied Probability
No flags	0	5%
`jurisdiction_missing` only	15	25%
`rate_mismatch` only	30	45%
`tax_missing` only	40	60%
`tax_on_nontaxable_item` only	35	50%
`gl_tax_diff_flag` only	25	40%
`tax_missing` + `rate_mismatch`	65	85%
`tax_missing` + `gl_tax_diff_flag`	70	90%
`tax_on_nontaxable_item` + `rate_mismatch`	60	80%
3 or more flags	85	95–99%

Jurisdiction logic is aligned with Florida surtaxes and total rates defined earlier.

4.4 Model D – Lease (ASC 842) Discrepancy Risk

Flags per lease:

- `schedule_to_GL_liability_diff_flag` – Year-end lease liability from schedule ≠ GL balance beyond tolerance
- `schedule_to_GL_ROU_diff_flag` – Year-end ROU asset from schedule ≠ GL balance beyond tolerance
- `missing_periods` – Missing schedule periods or Sequence_Error flags present
- `incorrect_opening_entry` – Opening entry differs from what the calculated schedule expects (simulated)
- `classification_flag` – Lease classification (operating vs finance) inconsistent with policy (simulated)

Weight matrix:

Condition	**Weight (risk points)**	**Implied Probability**
No flags	0	5%
Liability diff < 2% only	20	30%
ROU diff < 2% only	20	30%
`missing_periods` only	40	60%
`incorrect_opening_entry` only	50	70%
`classification_flag` only	45	65%
Liability diff ≥ 2% + `missing_periods`	70	90%
ROU diff ≥ 2% + `incorrect_opening_entry`	75	90–95%
3 or more flags	90	95–99%

The resulting `Lease_with_risk.csv` feeds the ASC 842 section of the dashboard and the lease-related JE suggestions.

5. How validator.py Uses scoring_rules.py

High-level flow:

1. Read base tables (`AP_Subledger.csv`, `Bank_Transactions.csv`, `Tax_Detail.csv`, `Lease_Schedule.csv`, `GL_Trial_Balance.csv`).

2. Compute foundational checks (e.g., Expected_Total, IP_Sum, sequence checks, jurisdiction rates).

3. For each domain (AP, Bank, Tax, Leases), call the appropriate scoring function from `scoring_rules.py` to:

    ○ Set boolean flags (missing_in_GL, tax_missing, etc.)

    ○ Compute `risk_score` and a `probability_band` label

4. Write enriched CSVs to `output/`:

    ○ `AP_with_risk.csv`
    ○ `Bank_with_risk.csv`
    ○ `Tax_with_risk.csv`
    ○ `Lease_with_risk.csv`

These enriched files are then used by the dashboard for visualization and review.

6. Streamlit Dashboard (dashboard.py)

The dashboard:

- Loads `output/*.csv`
- Allows filtering by risk score and vendor
- Provides leadership-friendly summaries and drill-downs

Key sections:

1. **Executive Close Status**
    ○ Counts of AP invoices and high-risk items across AP, bank, tax, and leases

2. **Year-End Close Health Overview**
    ○ Bar chart: high-risk items by functional area

3. **AP Reconciliation – Detail & Trend**
    ○ Risk distribution histogram

    - Root cause pie chart (missing in GL, mismatches, duplicates, unusual GL)
    - Table of high-risk invoices

4. **Cash and Bank Controls**
    - Average bank risk by vendor
    - Breakdown of no-match, duplicates, amount mismatches
    - Table of high-risk payments

5. **Sales & Use Tax – Florida Focus**
    - Tax risk by jurisdiction
    - Mix of rate mismatches, missing tax, GL variances
    - Table of tax records needing adjustment

6. **Lease Accounting – ASC 842**
    - Lease risk distribution
    - Missing periods and GL tie-out issues
    - Table of leases requiring review

7. **Journal Entry Preview & Export**
    - Suggested JEs for AP, tax, and leases derived from highest-risk items
    - CSV export for controller review and posting

Suggested journal entries are always **recommendations only** and must be reviewed and approved before posting in any production accounting system.

7. Running the Project Locally

From the project root:

1. Create and activate a virtual environment:

```
python -m venv .venv
source .venv/bin/activate  # macOS/Linux
# .venv\Scripts\activate   # Windows
```

2. Install requirements:

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

3. If starting from base data, run the validator:

```
python validator.py
```

This writes `AP_with_risk.csv`, `Bank_with_risk.csv`, `Tax_with_risk.csv`, and `Lease_with_risk.csv` to `output/`.

4. Run the dashboard:

```
python -m streamlit run dashboard.py
```

Then open the URL printed in your terminal (typically `http://localhost:8501`).

8. How Controllers and Billing Supervisors Use This

- **Controller**
  - Reviews high-risk items first
  - Validates whether proposed JEs are appropriate
  - Uses risk distributions and counts to judge overall close readiness
- **Billing Supervisor**
  - Investigates specific vendors or invoices identified as high risk
  - Coordinates corrections in AP and billing processes
- **Senior Leadership**
  - Sees a single, unified view of where risk and volume are concentrated
  - Gains confidence that year-end close is systematic, not purely manual

9. Future Enhancements

This framework can be extended to:

- Run on a schedule with a warehouse (BigQuery) backend
- Integrate with QuickBooks, NetSuite, or other ERPs
- Add natural language querying for non-technical users
- Incorporate ML-based anomaly detection on top of the rule-based layer
- Support multi-entity, multi-jurisdiction close in a consolidated environment

10. Intended Use

This repository is a demonstration project. Data is synthetic but realistic. Before using similar code in production:

- Align thresholds and weights with your materiality and control structure
- Review tax logic with tax specialists

- Review lease logic with technical accounting resources
- Confirm that suggested JEs follow your organization’s chart of accounts and policy

It is designed to show how an Accounting Associate can bridge detailed reconciliations, internal controls, and lightweight data engineering to create a “close command center” for year-end work.