Transforming online retail data using a Data Pipeline

Agenda

- Context & Goals
- Architecture
- Pipeline Execution
- Data Quality
- Model & Insights
- Q&A ?

A MOSAKU, DATA ENGINEERING CANDIDATE | 14 NOV 2025

CHALLENGE & EVALUATION LENS

Assessment brief: Ingestion → Wrangling → Clean → Modelling → Documenting

Expectation reminder

Evaluation Criteria	Evidence in Delivery
Code quality	Modular Python package under src/pipeline, logging + type-safe configs
Data modelling	Dimensional star schema, parquet marts, sql/create_tables.sql DDL
Data quality approach	Automated validation checks, documented thresholds, pipeline.log artifacts
Pipeline robustness	Error handling, stage-wise logging, idempotent outputs
Documentation	README, data model notes, data dictionary, architecture diagram, this note too
Communication	Presentation narrative, quick notebook walkthrough, explanation of trade-offs
Problem-solving	Business-rule enforcement, cleansing strategy, possible enhancements

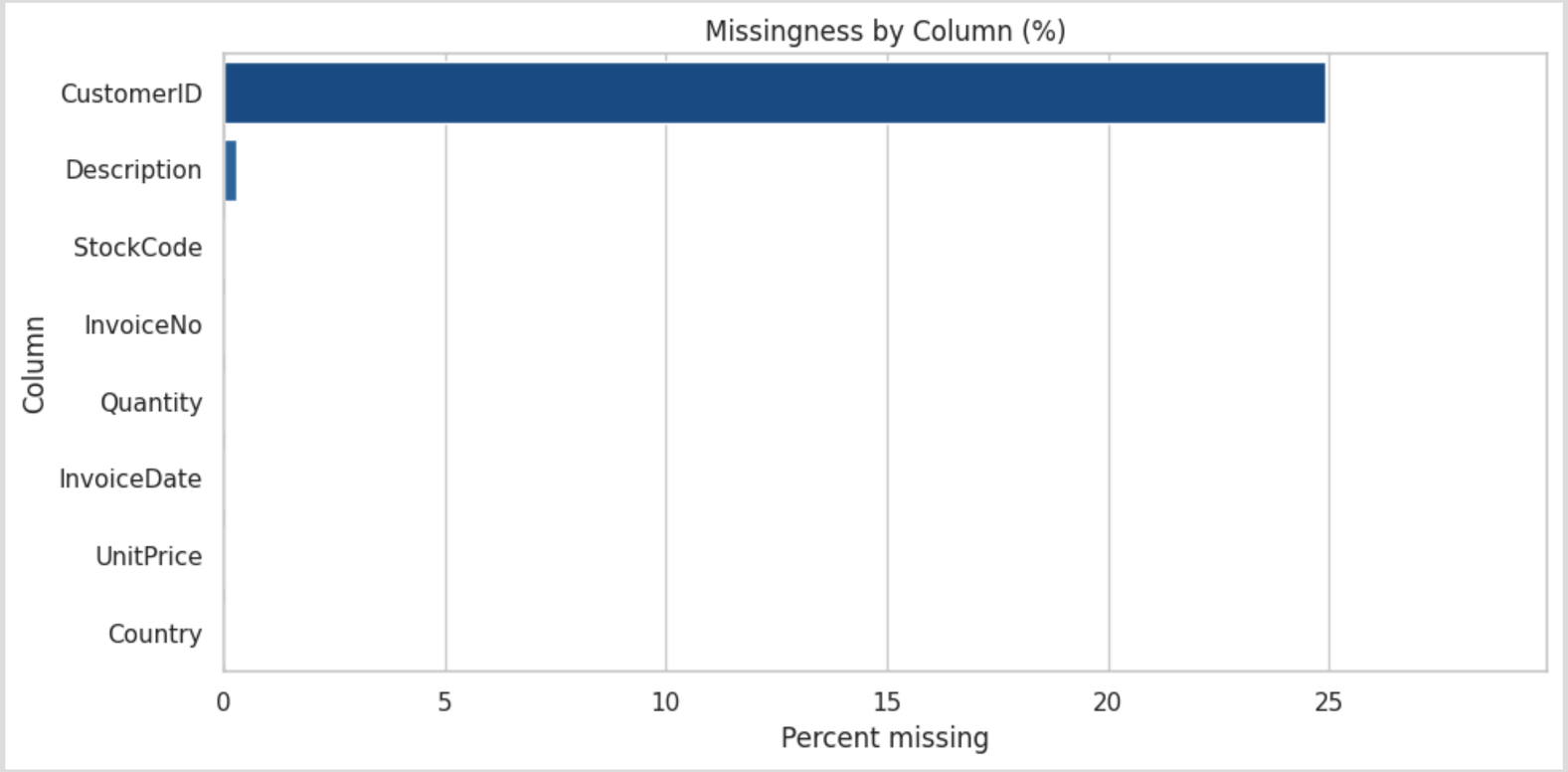
DATASET & PROFILING SNAPSHOT

Source: UCI Online Retail (01-Dec-2010 to 09-Dec-2011, 541k invoice lines, 8 fields).

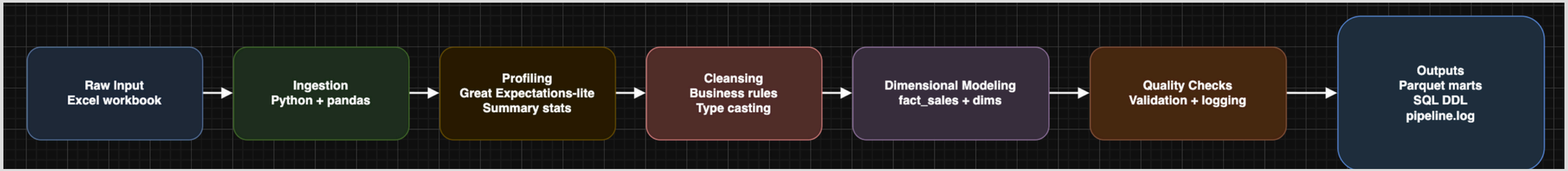
Business rules enforced:

- Drop rows with missing customers
- filter non-positive quantities/prices
- deduplicate invoice lines
- normalise invoice dates to datetime.

	rule	records
0	Quantity < 0	10624
1	UnitPrice <= 0	2517
2	Missing CustomerID	135080
3	Duplicate InvoiceNo+StockCode+InvoiceDate	10677



ARCHITECTURE OVERVIEW

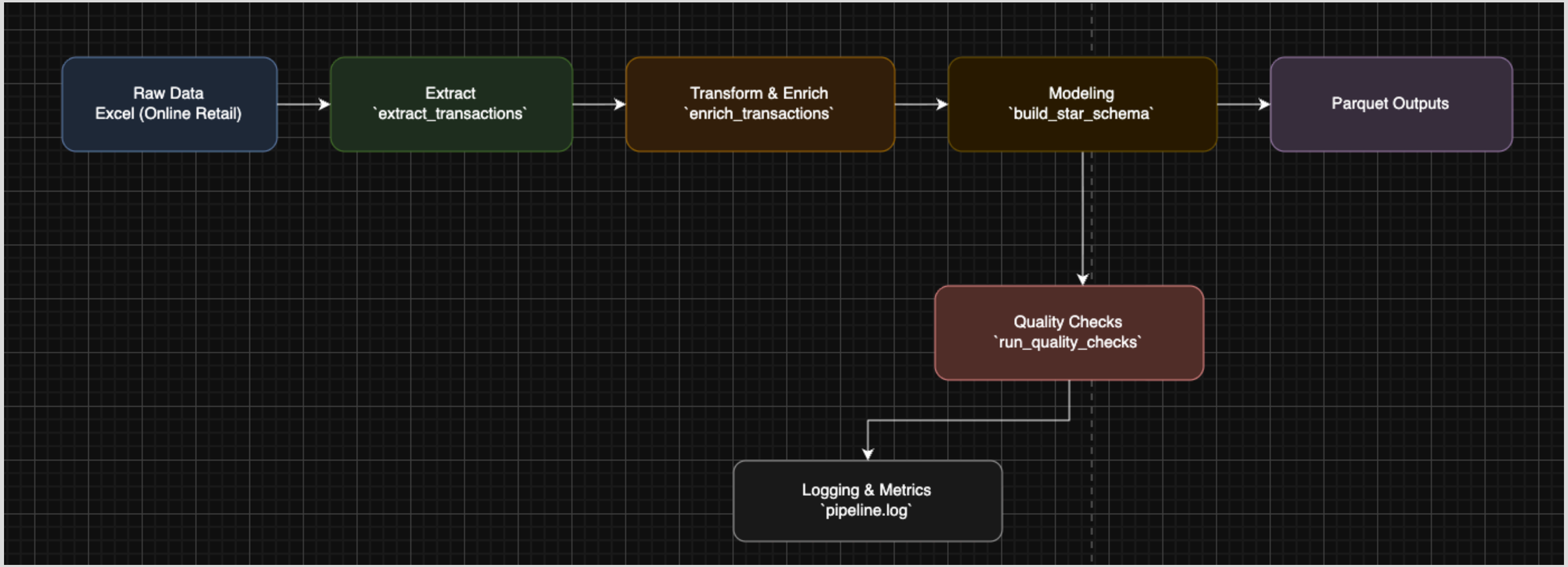


Tech stack: Python 3.11, pandas, standard logging,

Key trade-off: Full ETL Pipeline rather than an ELT, all to minimise setup; SQL DDL provided for warehouse alternative, DuckDB/dbt noted as future enhancement.

Documentation pointer: All docs and SQL assets sit in docs/ and sql/ .

PIPELINE ORCHESTRATION

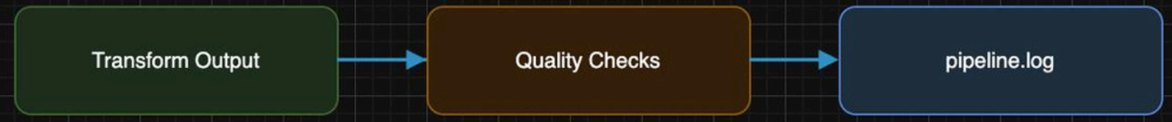


DATA QUALITY FRAMEWORK

Checks implemented:

- Python raises exceptions on fail
- Logs them to pipeline.log with stage + severity
- Warning counts captured for reporting.

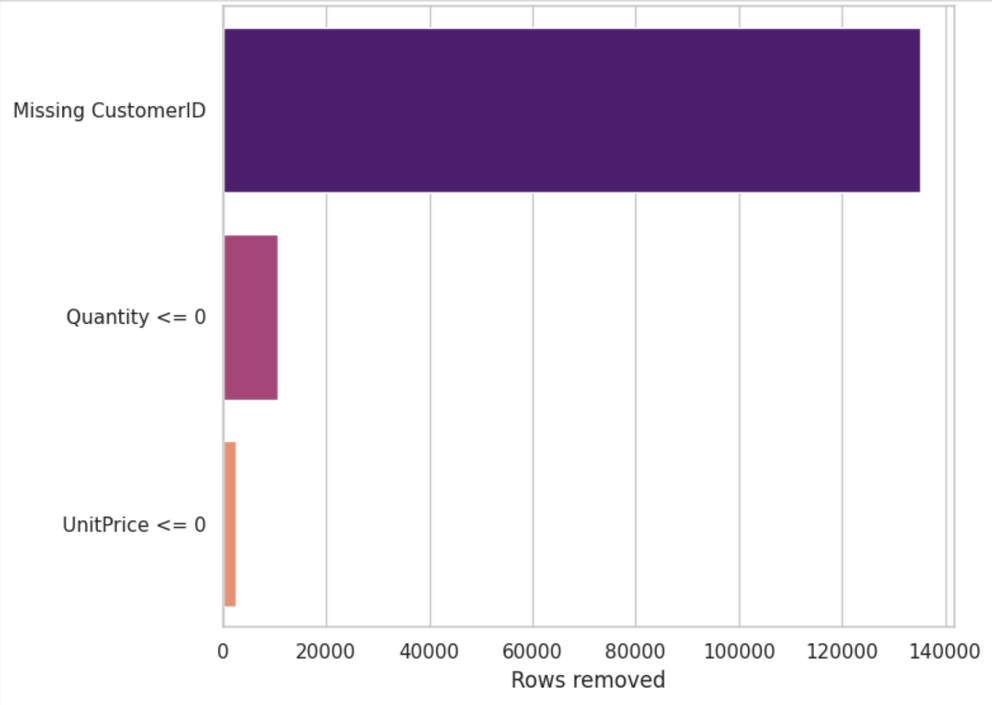
Check	Rule	Severity	Action	Evidence
Quantity validity	Quantity > 0	Fail	Raise exception, drop row	pipeline.log / quality report
Price validity	UnitPrice > 0	Fail	Raise exception, drop row	pipeline.log
Customer completeness	CustomerID not null	Fail	Drop row, log count	pipeline.log
Referential integrity	Fact rows join to dims	Fail	Abort load	fact vs dim counts
Row reconciliation	Raw vs cleaned delta	Warn	Log variance	pipeline.log summary

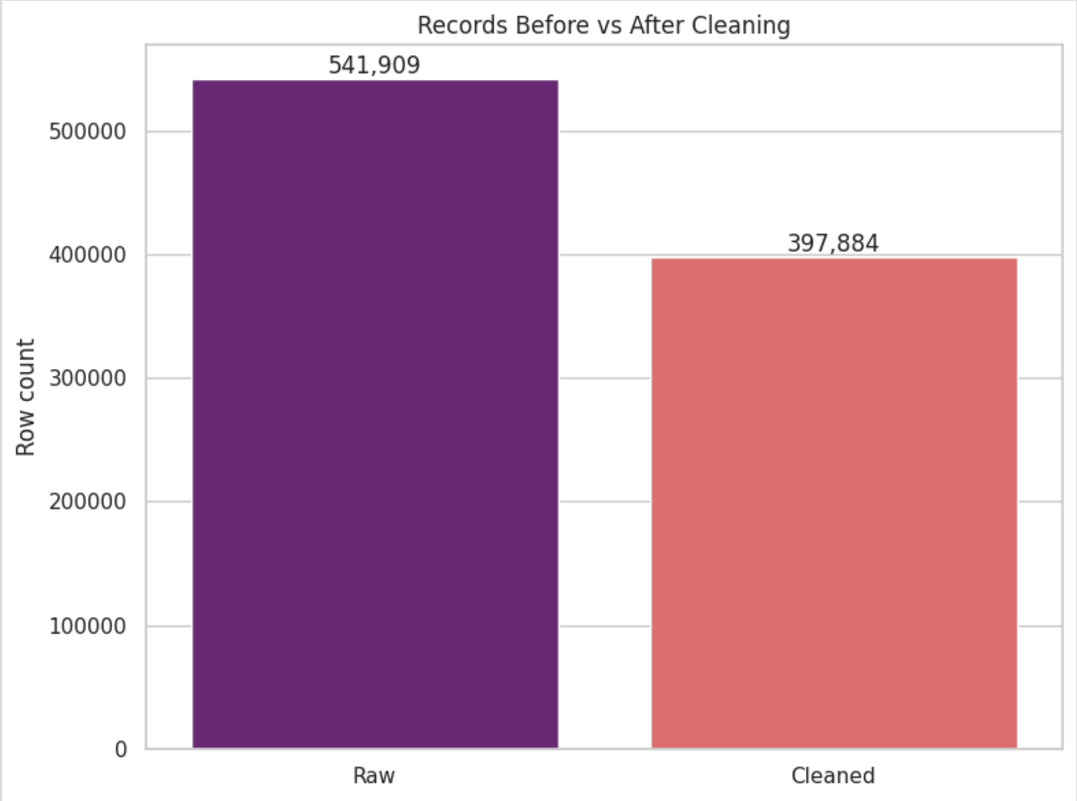


TRANSFORMATION & CLEANSING RESULTS

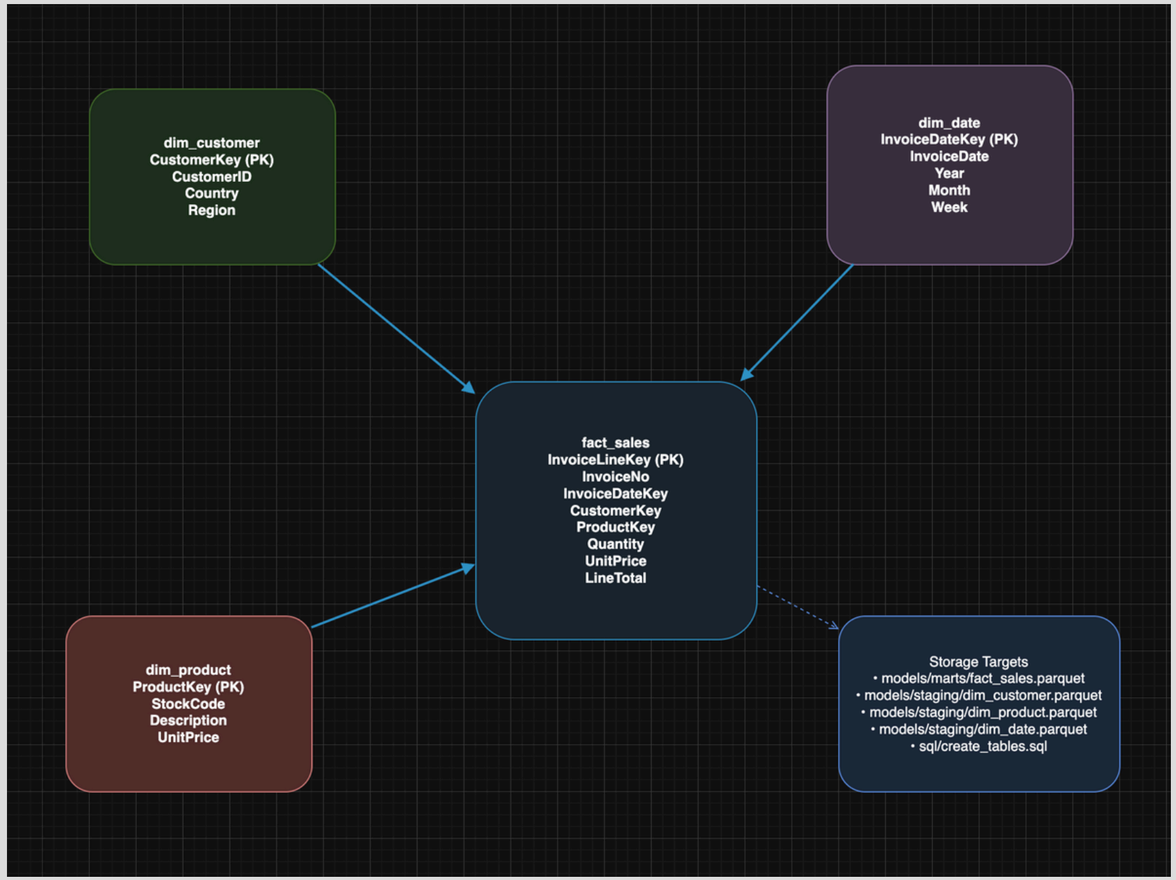
Headline metrics:
Raw rows 541,909 → Cleaned rows 397,884 (−26.6%);

<u>Rows removed</u>:
135,080 missing customers,
8,905 non-positive quantities (after customer filter)
44 non-positive prices
distinct customers retained: dim_customer has 4,346 rows





DIMENSIONAL MODEL & STORAGE



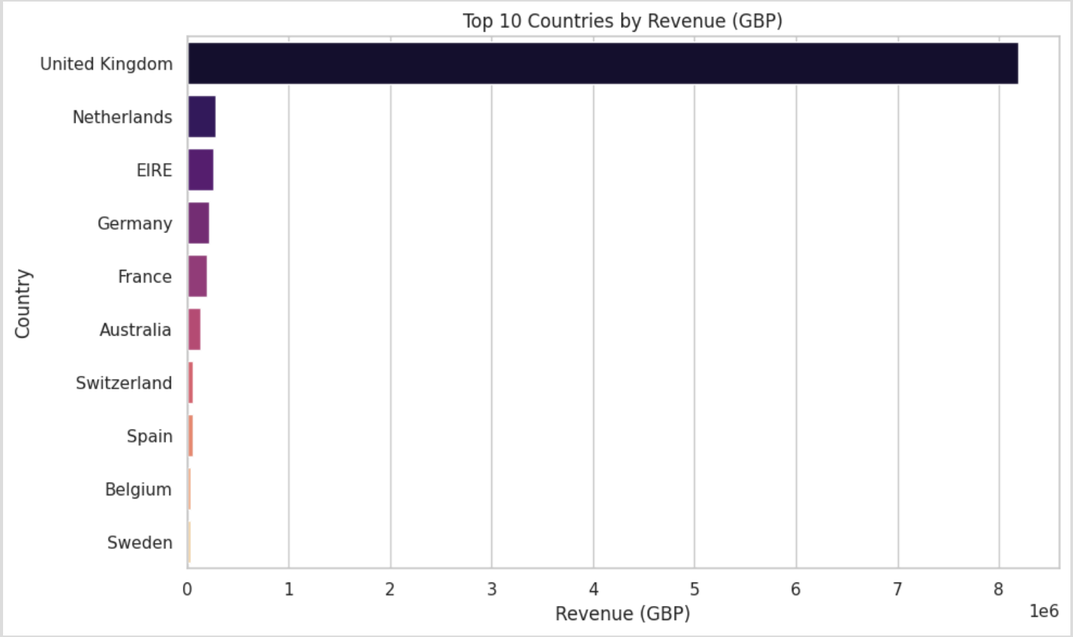
Star schema: Our ‘Fact_sales` table at linking to `Dim_customer`, `Dim_product`, and `Dim_date`.

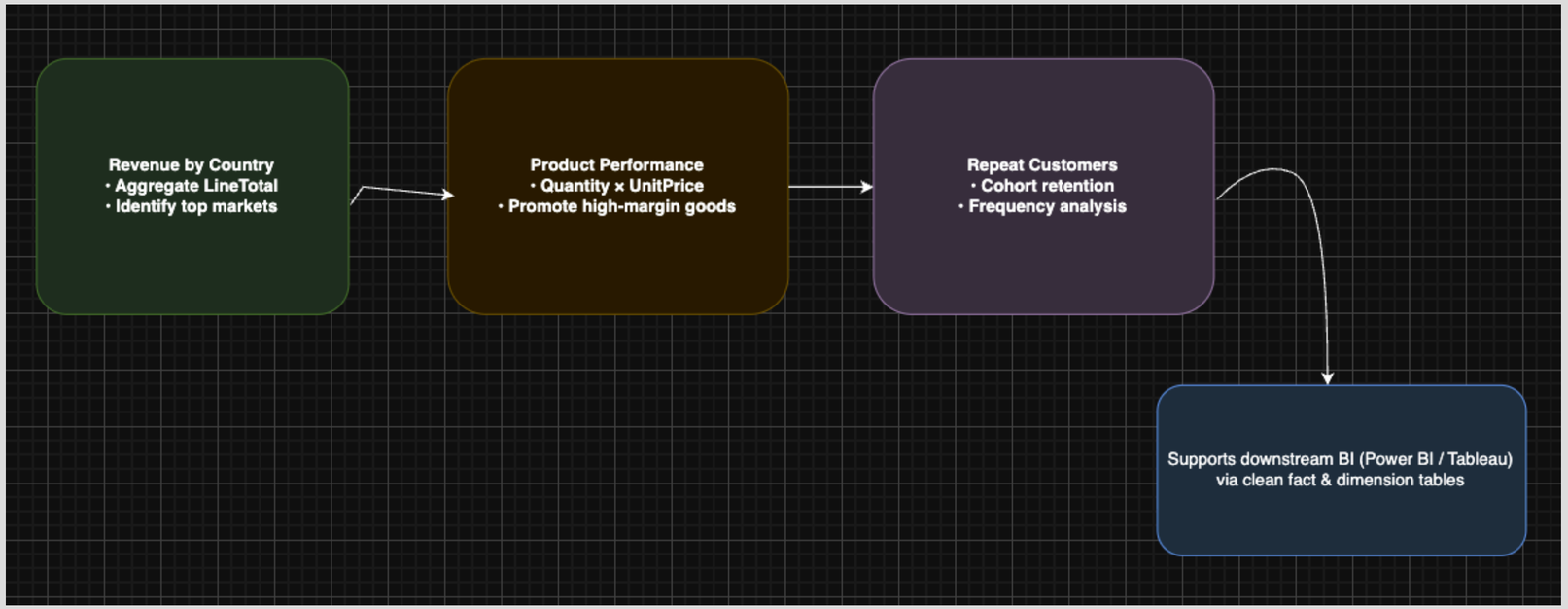
Key columns:
Fact = InvoiceNo, InvoiceDateKey, CustomerKey, ProductKey, Quantity, UnitPrice,

LineTotal:
Dimensions store descriptive attributes with surrogate keys.

ANALYTICAL USE CASES

Use cases unlocked: Revenue by country (GBP), product performance trends (quantity × unit price), repeatable customer frequency/cohort analysis.
BI readiness: Dimensions + fact enable Power BI dashboards with minimal modeling.
Evaluation tie: Demonstrates schema suitability for downstream analytics.





DEMO EVIDENCE

```
chrismo@chrismos-MacBook-Pro CaseStudy % source .venv/bin/activate
(.venv) chrismo@chrismos-MacBook-Pro CaseStudy % python -m src.pipeline.run
<frozen runpy>:128: RuntimeWarning: 'src.pipeline.run' found in sys.modules after import of package 'src.pipeline', but prior to execution of 'src.pipeline.run'
; this may result in unpredictable behaviour
2025-11-12 13:51:46,493 [INFO] __main__ - Starting Online Retail pipeline
2025-11-12 13:51:46,494 [INFO] __main__ - Configuration: input=data/Online Retail.xlsx staging=models/staging marts=models/marts min_quantity=1 min_unit_price=0
.01 chunk_size=None
2025-11-12 13:52:03,406 [INFO] __main__ - Extracted 541909 rows from raw dataset
2025-11-12 13:52:03,689 [INFO] __main__ - Transformation stats: {'initial_rows': 541909, 'missing_customer': 135080, 'bad_quantity': 8905, 'bad_unit_price': 44,
 'invalid_invoice_date': 0, 'rows_retained': 397880}
2025-11-12 13:52:05,243 [INFO] __main__ - Table fact_sales has 397880 rows
2025-11-12 13:52:05,244 [INFO] __main__ - Table dim_customer has 4346 rows
2025-11-12 13:52:05,244 [INFO] __main__ - Table dim_product has 8879 rows
2025-11-12 13:52:05,244 [INFO] __main__ - Table dim_date has 17282 rows
2025-11-12 13:52:05,556 [INFO] __main__ - Persisted tables to disk: {'fact_sales': PosixPath('models/marts/fact_sales.parquet'), 'dim_customer': PosixPath('mode
ls/staging/dim_customer.parquet'), 'dim_product': PosixPath('models/staging/dim_product.parquet'), 'dim_date': PosixPath('models/staging/dim_date.parquet')}
2025-11-12 13:52:05,628 [INFO] __main__ - Quality check FactSalesQuantityPositive: True ({'rows_with_bad_quantity': 0})
2025-11-12 13:52:05,628 [INFO] __main__ - Quality check FactSalesUnitPricePositive: True ({'rows_with_bad_price': 0})
2025-11-12 13:52:05,628 [INFO] __main__ - Quality check FactSalesCustomerExists: True ({'missing_customer_keys': 0})
2025-11-12 13:52:05,628 [INFO] __main__ - Quality check FactSalesProductExists: True ({'missing_product_keys': 0})
2025-11-12 13:52:05,628 [INFO] __main__ - Quality check FactSalesInvoiceLineUnique: True ({'duplicate_lines': 0})
(.venv) chrismo@chrismos-MacBook-Pro CaseStudy %
```

ASSUMPTIONS, RISKS, FUTURE WORK

What we assumed	Why it’s risky	What to do next
We drop any rows that don’t have a CustomerID.	We lose the sales tied to those orders.	Reconnect whatever source we exported from or park them in an “unknown customer” group until we know who they are.
We keep only the latest UnitPrice for each product.	Price changes over time disappear.	Track price history or add a slowly changing dimension so we can see older prices.
We process one big file each run.	There’s no way to load data in chunks or run incrementally.	Add orchestration (like Airflow) and split loads into batches.
We built the first set of quality rules only.	Other data issues might slip by.	Expand the rule set, ideally with a tool like Great Expectations.

Q & A