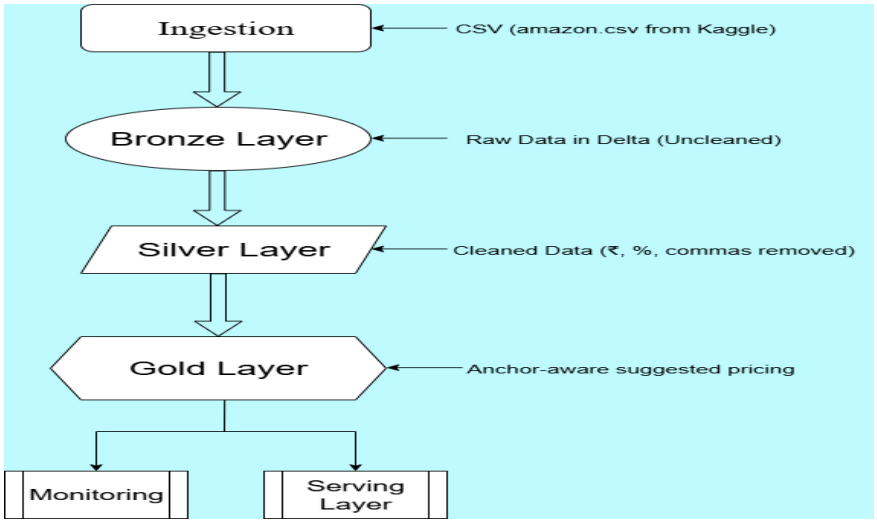
# Dynamic Product Pricing – System Design & Prototype

---

## High-Level Architecture Diagram




## Core Components

- **Data Source:** CSV (amazon.csv) with Amazon product data (ratings, products, Prices...). https://www.kaggle.com/datasets/karkavelrajaj/amazon-sales-dataset
- Ingested into Databricks using Spark's read.csv.
- Could extend to batch or streaming ingestion in production (e.g., Kafka for clickstream).

## Storage Design

- Used Delta Lake tables for Bronze, Silver, Gold.
- **Partitioning strategy:** would partition by category_simple if scaling to billions of rows.
- Bronze: raw schema as-is.
- Silver: cleaned schema (numeric prices, ratings).
- Gold: derived suggested pricing.

## Feature Engineering & Transformation

- Created **popularity_score = rating × rating_count**.
- Defined **anchors** = top 10% products by popularity.
- Computed **anchor_median_price** per category.
- Suggested price = **0.9 * anchor_median_price + 0.1 * actual_price**.

## Experimentation Support

- Pricing formula is configurable (e.g., 90/10 could try 80/20, 70/30).
- Logs could be added for A/B testing different weights.
- With more time: integrate ML-based models to learn optimal weights.

### Serving Layer

- Gold table (**amazon_gold_pricing**) stored in Delta Lake.
- Accessible via SQL queries, BI dashboards, or exposed as API endpoints.
- Comparison view (**actual_price vs suggested_price**) used for demo.

### Monitoring & Alerting

- Null counts for critical columns (**actual_price, discounted_price, discount_percentage, rating, rating_count**).
- Sanity checks: prices > 0.
- With more time: integrate alerts into Databricks Jobs.

### Databricks Integration

- Used Databricks Notebooks + Spark + Delta Lake.
- Cluster: serverless free edition.
- Pipeline fully runs inside Databricks environment.

### Reliability & Scalability

- Delta Lake ensures ACID transactions and schema evolution.
- Partitioning by category improves query speed.
- Can scale horizontally on Databricks clusters.

### AI-Assisted Development

- Used ChatGPT for:
- regexp_replace, try_cast
- anchor product logic
- Drafting Gold pipeline with joins
- Creating this PDF

### Future Improvements (if more time)

- Train an ML model (regression or gradient boosting) for price prediction.
- Add real-time clickstream ingestion (Kafka -> Spark Streaming).
- Automate pipeline with Databricks Jobs + CI/CD.
- Add alerts + dashboards for monitoring.

### Trade-offs

- Chose rule-based anchor pricing (simple, explainable, fast) instead of ML (more accurate but slower).
- Simplified category hierarchy to top-level (Electronics, Computers, etc.) to fix messy data joins.
- Used a single dataset (Amazon reviews) instead of multiple real-world data sources.