**Dataset Overview**

The application uses a synthetic dataset that simulates a partner ecosystem with hierarchical relationships and multiple performance metrics. This synthetic data is generated using the functions in data.py when the application starts, rather than using real external data.

**1. Partner Data Structure**

**Basic Partner Information**

* **partner\_id**: Unique identifier for each partner
* **name**: Partner name (e.g., "Partner 1", "Root Distributor")
* **level**: Partner tier/category (Distributor, Agent, or Ambassador)
* **parent\_id**: ID of the parent partner (establishes hierarchy)
* **join\_date**: When the partner joined the program
* **status**: Partner status (Active, Pending, Inactive, or Premium)
* **total\_revenue**: Aggregate revenue associated with this partner

**Digital & Social KPIs**

* **posts**: Number of social media/marketing posts created by partner
* **shares**: Number of content shares/reposts from the partner
* **sentiment**: Sentiment score (-1.0 to 1.0) measuring customer perception
* **advocacy\_score**: Score (1-100) measuring partner's brand advocacy
* **engagement**: Overall engagement score (1-100)

**Time-Series Data**

* **Sales Data**: partner\_id, date, revenue, transaction\_id, product
* **Activity Data**: partner\_id, date, activity\_type, duration\_minutes
* **Social Data**: partner\_id, date, posts, shares, sentiment, advocacy\_score, reviews

**2. Attribute Definitions**

**Partner Levels**

Represents the partner's tier in your channel program hierarchy:

* **Distributor**: Top-level partners who may have their own network of sub-partners
* **Agent**: Mid-tier partners who sell/represent your products
* **Ambassador**: Entry-level or advocacy-focused partners

**Social/Digital Metrics**

* **posts**: Marketing content published by the partner mentioning your brand/products
* **shares**: How often the partner redistributes your content or has their content shared
* **sentiment**: Analysis of customer feedback/comments associated with this partner (-1 = negative, 0 = neutral, 1 = positive)
* **advocacy\_score**: Quantitative measure of how actively the partner promotes your brand
* **reviews**: Customer reviews submitted through this partner's channels

**Activity Types**

Various partner engagement activities tracked:

* Login
* Call
* Meeting
* Demo
* Training
* Review

**3. Real-World Data Ingestion**

In a real-world scenario, this data would come from multiple sources. Here's how you might approach data ingestion:

**3.1 Data Sources**

The data would likely come from several systems:

* **CRM System**: Partner profile data, relationship hierarchy, status updates
* **ERP/Sales System**: Revenue and transaction data
* **Partner Portal**: Login, training, and engagement activity
* **Social Media APIs**: Posts, shares, and engagement metrics
* **Sentiment Analysis Tools**: Processed feedback, reviews, and sentiment scoring
* **Marketing Automation**: Campaign participation and content sharing

**3.2 Ingestion Methods**

You would typically implement the following approaches:

**API Integrations**

* Connect directly to CRM APIs (e.g., Salesforce, Microsoft Dynamics)
* Integrate with social media platforms' APIs
* Use sentiment analysis tools' APIs (e.g., Azure Text Analytics, AWS Comprehend)

**ETL Processes**

* Create scheduled ETL jobs to extract data from various sources
* Transform and normalize the data for consistency
* Load into a central data warehouse or data lake

**Event-Based Integration**

* Use webhooks to capture real-time events (e.g., new reviews, sales)
* Process these events through a message queue (e.g., Kafka, RabbitMQ)
* Update the dashboard data in near real-time

**3.3 Implementation Example**

# Example of a real-world data connector

def ingest\_partner\_data():

# 1. Partner profile data from CRM

partner\_data = crm\_connector.get\_partners()

# 2. Sales data from ERP

sales\_data = erp\_connector.get\_sales\_transactions(date\_range=last\_90\_days)

# 3. Activity data from partner portal

activity\_data = portal\_connector.get\_partner\_activities(date\_range=last\_90\_days)

# 4. Social media data

social\_data = []

for platform in ['twitter', 'linkedin', 'facebook']:

social\_data.extend(social\_connector.get\_metrics(platform=platform))

# 5. Sentiment analysis

reviews = review\_connector.get\_reviews(date\_range=last\_90\_days)

sentiment\_results = sentiment\_analyzer.analyze\_text([r.text for r in reviews])

# 6. Transform and combine data

transformed\_data = data\_transformer.normalize\_and\_combine(

partner\_data, sales\_data, activity\_data, social\_data, sentiment\_results

)

# 7. Load to database

database.upsert\_data(transformed\_data)

**4. Real-time vs. Batch Processing**

Depending on your needs:

* **Batch Processing**: Schedule daily/weekly updates for non-time-critical data
* **Real-time Processing**: Implement streaming for critical metrics that need immediate visibility

**5. Data Storage Considerations**

For a production implementation:

**Database Options**

* **Relational Database** (e.g., PostgreSQL) for structured relationship data
* **Time Series Database** (e.g., InfluxDB, TimescaleDB) for metrics over time
* **Data Warehouse** (e.g., Snowflake, BigQuery) for analytics at scale

**Schema Design**

* Partner dimension table with hierarchical relationship
* Fact tables for sales, activities, and social metrics
* Denormalized views for dashboard performance

**6. Extending the Application**

To make this POC production-ready, you might:

* Replace the generate\_partners(), generate\_sales(), etc. functions with real data connectors
* Implement incremental data loading rather than regenerating everything
* Add authentication and role-based access control
* Implement data refresh scheduling
* Add data validation and error handling

**Conclusion**

This RPA Accelerator provides a flexible framework that can be extended to ingest real data from your specific partner ecosystem while maintaining the same visualization and analysis capabilities.