Here’s the **modified OPAIR Framework** tailored to generate all sample data into a single .py file instead of multiple modular files:

**Single-File Industry Data Generator – Built using OPAIR**

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Objective** | To enable rapid creation of realistic, industry-specific sample datasets through a **single Python file**, simplifying deployment for prototyping, demos, or teaching—without using sensitive data. |
| **Principles** |  |
| *Simplicity Over Modularity* | All logic is consolidated into one file for ease of sharing, deployment, and experimentation. |
| *Integrated Data Simulation* | Sales, inventory, and master data are generated inline using defined structures. |
| *Realism via Probability* | Controlled randomness simulates customer types, behaviors, and transaction flows. |
| *Configurable Within Code* | Parameters (volume, date range, entities) are set in top-level config blocks. |
| *Multi-Format Export in One File* | Supports export to CSV or Excel directly from the script, with no external dependencies. |

### **A – Actions Enabled by This Framework**

|  |  |
| --- | --- |
| **Action** | **Outcome** |
| Simulate Industry Transactions | Generate Sales, Claims, Appointments, or Visits inline. |
| Build Entity Relationships | Create master-detail logic directly within a single file. |
| Produce Time-Series Data | Control frequency and volume of events per time unit. |
| Apply Probability Models | Set ratios for VIP/Regular customers, online/in-store, etc. |
| Output Multi-Entity Dataset | Write multiple tables (e.g., SalesHeader, SalesDetail, ProductMaster) from one run. |

### **I – Inputs to Activate the Framework**

|  |  |
| --- | --- |
| **Input Type** | **Description** |
| industry\_name | e.g., "Retail", "Healthcare", "Banking" |
| entities\_to\_simulate | List of tables: e.g., ["SalesHeader", "SalesDetail", "Inventory"] |
| date\_range | Start and end dates: e.g., ("2020-01-01", "2025-12-31") |
| volume\_per\_entity | Dictionary specifying rows: e.g., {"SalesHeader": 1000, "ProductMaster": 20} |
| probability\_rules | e.g., {"customer\_type": {"VIP": 0.3, "Regular": 0.7}, "channel": {"Online": 0.2, "In-store": 0.8}} |
| output\_format | "CSV" or "Excel" (stored to disk from within the same script) |

### **R – Real-World Example**

**Scenario**: A Retail BI consultant needs mock data to test dashboards for a 5-store fashion chain.

**How the Framework Was Used**:

* Edited a single .py script to set industry\_name = "Retail" and selected entities.
* Created SalesHeader, SalesDetail, ProductMaster, and Inventory data.
* Applied rules: 70% Regular, 30% VIP; 80% In-store, 20% Online.
* Generated 3,000 rows of transactions spanning 2 years.
* Exported all tables into one Excel file using pandas.ExcelWriter.

**Result**: A complete, high-quality test dataset was created in under 2 minutes using only one file—ideal for sharing with clients or students.

### **🔧 Single .py File Structure (Code Overview)**

|  |  |
| --- | --- |
| **Section** | **Purpose** |
| config\_block | User-editable section to set industry, entities, volume, and probability rules |
| generate\_data() | Main function that simulates all requested entities and relationships |
| apply\_probability() | Encodes weighted random logic |
| export\_data() | Handles CSV or Excel output inline using Pandas |
| main() | Entry point to control the workflow |