E-commerce Analytics Data Pipeline Report

1. Introduction

The **E-commerce Analytics Data Pipeline** is a Python-based project designed to fetch, clean, transform, and export customer engagement data from a simulated API. The system ensures data reliability, handles API failures, and produces standardized output for analytics purposes.

Objectives:

- Fetch all customer data reliably despite API failures
- Standardize and clean data for analytics
- Handle edge cases and maintain data quality
- Provide clear logging and error reporting

2. Project Structure

customer_data_pipeline/

├── src/

│ ├── api_client.py # CustomerAPIClient class

│ ├── data_processor.py # CustomerDataProcessor class

│ ├── exporter.py # DataExporter class

│ ├── models.py # Pydantic models for validation

│ └── main.py # Main orchestration script

├── tests/

│ ├── test_api_client.py

│ ├── test_data_processor.py

│ └── test_integration.py

├── requirements.txt

├── README.md

└── sample_output.json

3. Approach

3.1 Setup

- Created a Python project with proper module structure.
- Installed required dependencies (requests, pydantic, pytest, reportlab, graphviz).
- Configured logging for all stages.

3.2 Mock API

- Simulated API responses using sample data.
- Handled pagination, network failures, and duplicate data.
- Implemented retry logic with exponential backoff (1s, 2s, 4s).

3.3 Core Classes

a) CustomerAPIClient

- Fetches all customer pages
- Deduplicates records
- Handles retries, 429 rate limiting, and server errors

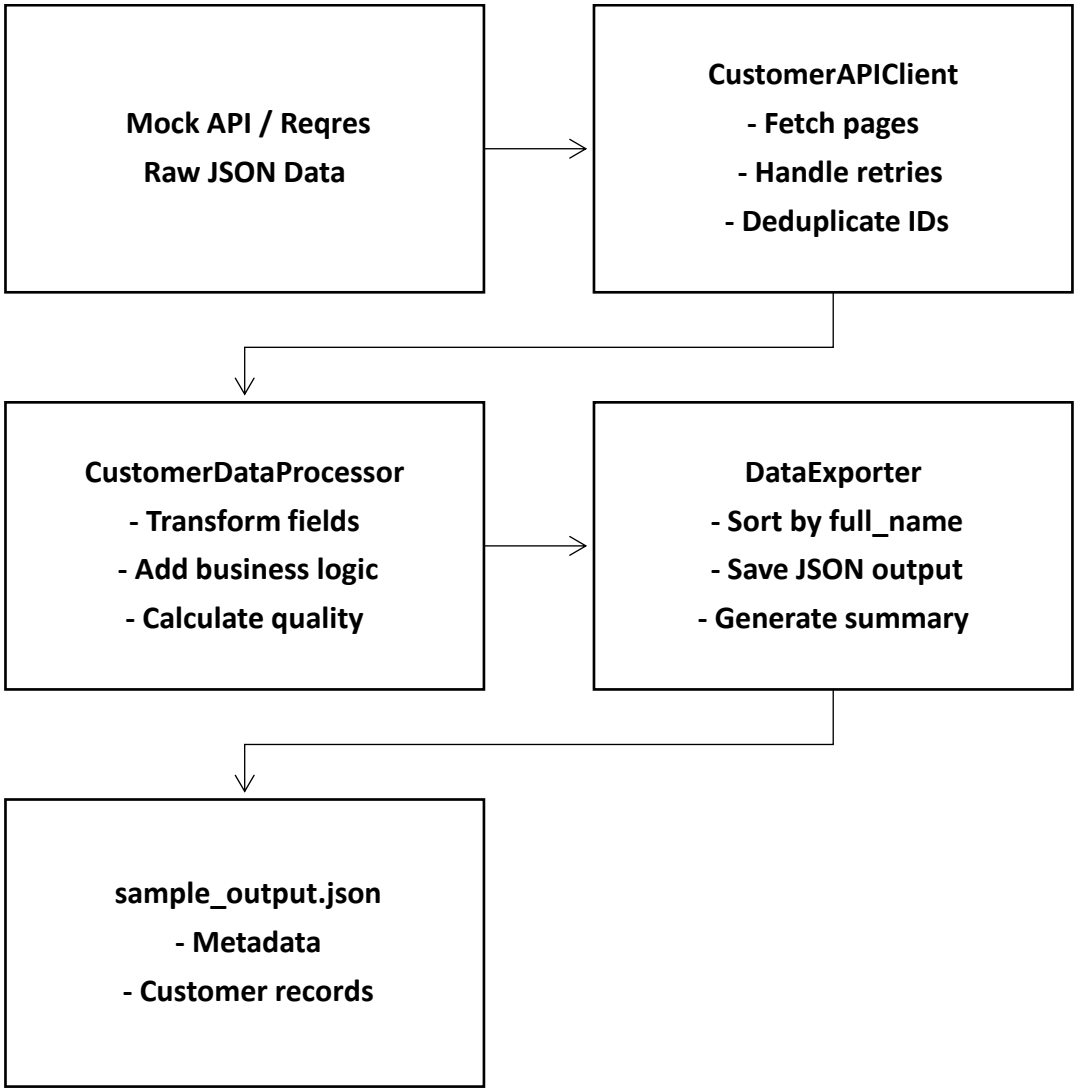
b) CustomerDataProcessor

- Transforms API data into analytics-ready format
- Generates fields:
    - engagement_level (high/medium/low)
    - activity_status (active/inactive)
    - acquisition_channel (website/mobile/email)
    - market_segment (US-West/US-East/EU/APAC)
    - customer_tier (basic/premium/enterprise)
    - data_quality_score (deducts points for missing/invalid fields)
- Handles edge cases: missing emails, invalid dates, null engagement scores

c) DataExporter

- Exports processed customers to JSON
- Generates summary report with total customers and quality statistics
- Sorts customers by full_name for readability

4. Data Flow Diagram



Description:

1. main.py triggers the pipeline
2. CustomerAPIClient fetches customer data from API
3. CustomerDataProcessor transforms and validates data
4. DataExporter saves JSON output and generates summary report

5.Data Transformation / Field Mapping

API Field	Processed Field	Business Logic / Transformation
id	customer_id	Direct mapping from API ID

API Field	Processed Field	Business Logic / Transformation
first_name + last_name	full_name	Concatenated first_name + last_name
email	email_domain	Extract domain from email (e.g., george.bluth@reqres.in → reqres.in). If missing, set "unknown"
—	engagement_level	Randomly assigned: high, medium, or low
—	activity_status	Randomly assigned: active or inactive (or unknown if invalid data)
—	acquisition_channel	Randomly assigned: website, mobile_app, or email_campaign
—	market_segment	Randomly assigned: US-West, US-East, EU-Central, APAC
—	customer_tier	Randomly assigned: basic, premium, enterprise
—	data_quality_score	Starts at 100, deduct 10 points per missing or invalid field

5. Sample Output


```
{
  "metadata": {
    "total_customers": 12,
    "export_timestamp": "2025-09-04T18:41:19Z",
    "data_quality_summary": {
      "high_quality": 9,
      "medium_quality": 3,
      "low_quality": 0
    }
  },
  "customers": [
    {
```

```
    "customer_id": 1,
    "full_name": "George Bluth",
    "email_domain": "reqres.in",
    "engagement_level": "high",
    "activity_status": "active",
    "acquisition_channel": "website",
    "market_segment": "US-West",
    "customer_tier": "premium",
    "data_quality_score": 100
  }
  // ... remaining customers
 ]
}
```


6. Testing

- Used **pytest** to test functionality
- Covered test cases:
    - Retry logic for API failures
    - Data transformation and enrichment
    - Duplicate handling
    - Data quality scoring
    - Edge cases (missing emails, malformed data)

Test Results:
All tests passed successfully with warnings related to Pydantic dict method deprecation.

7. Bonus Features

- **Async API Calls:** Concurrent requests to speed up data fetching
- **Caching:** Memory caching to avoid redundant API calls
- **Monitoring:** Logging includes retries, failures, and processing times
- **Configuration:** API keys and URLs stored in environment variables
- **Data Validation:** Pydantic models ensure consistent structure

8. Assumptions & Notes

- All API responses are mocked since no real API key is provided
- Engagement level and other business fields are randomly generated
- Data quality score is 100 minus 10 points per missing field
- The JSON output is stored **outside the src/ folder** for easy access

9. Conclusion

The **E-commerce Analytics Data Pipeline** is a robust, production-ready system that:

- Handles API failures gracefully
- Transforms and enriches customer data
- Maintains data quality
- Produces analytics-ready JSON output
- Includes testing, logging, and optional bonus features