**Technical Documentation for Data Quality Visualisation System**

### **Introduction**

This document provides a brief and simplified example of technical documentation for a specific data management project, utilising the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology. CRISP-DM is a structured approach to data mining that ensures all phases of a project from understanding business objectives to deploying actionable insights are effectively addressed. This methodology is widely adopted due to its flexibility and adaptability across industries. The following sections outline how CRISP-DM principles are applied in this project to manage data loading, profiling, analysis, and visualisation processes efficiently.

### **1. Business Understanding**

The objective of this system is to efficiently manage data loading, profiling, analysis, and visualisation processes for employee, procurement, and inventory data using a combination of Python tools and open-source libraries. The system facilitates data quality validation, control, and reporting aligned with business objectives.

### **2. Data Understanding**

* **Data Sources**: SQL tables for employee, procurement, and inventory data.
* **Data Structure**: Includes descriptive metadata such as file type, size, and data types.
* **Compatibility**: Data loading technology options include Pandas, Polars, and scalable frameworks like PySpark.

### **3. Data Preparation**

* **Data Loader**: Utilises Python libraries for managing and transforming data into DataFrames for analysis and visualisation.
  + Creation of dummy data for testing.
  + Integration with SQL databases.
* **Configuration**:
  + Excel documents to manage file/database secrets.
  + Control tables to log application processes and track adoption rates (e.g., JSON format logs).

### **4. Modelling**

* **Descriptive Analysis**:
  + Profiling data structure, file type, size, and data types.
  + Quality metrics and column-level analysis using tools like Great Expectations and ydata-profiling.
* **Key Features**:
  + Bronze profiling on initial data upload to identify unassessed files.
  + Visualisation of row-level, column-level, and table-level information.

### 

### **5. Evaluation**

#### **Feature Set Overview**

| **Feature** | **Trigger** | **Method** | **Description** | **Compatibility** | **Value** |
| --- | --- | --- | --- | --- | --- |
| Data Loader | Demonstration | SQL Table Connection | Dummy data creation for employee and procurement data. | Not Applicable |  |
| Data Loader | Data Handling | Pandas / Polars | Analysis and visualisation support with scalable options like PySpark. | Technology scalability enabled | Maintained by open-source community with best practices. |
| App Information | Configuration | File/database secrets | Excel-based configuration management for technical integration. | - |  |
| Analysis | Descriptive | Bespoke Code / Libraries | Metadata analysis for data structure and quality metrics. | Open-source tools (e.g., Python) | Scalability and best practices with community-maintained tools. |
| App Analysis | Bronze Profiling | Initial Upload Profiling | Identifies files not previously assessed. | - | Bespoke, scalable with Python tools. |
| Visualisation | Data Overview | Quality Metrics Visualisation | Provides insights into data quality and structure. | - |  |

### 

### **6. Deployment**

#### **System Capabilities**

1. **Data Loading**:
   * Handles SQL integrations and dummy data creation.
   * Logs run information in JSON format for traceability.
2. **Data Analysis**:
   * Column-level, row-level, and table-level insights.
   * Quality metrics evaluation.
3. **Visualisation**:
   * Data profiling and descriptive statistics.
   * Outputs for failed data and quality reports.
4. **Rule Management**:
   * Visualised outputs for data rule management and quality metrics.

#### **Deployment Steps**

1. Install required Python libraries (e.g., Pandas, Polars, Great Expectations).
2. Configure file/database secrets using Excel documents.
3. Set up control tables for process tracking.
4. Deploy analysis scripts for descriptive profiling.
5. Integrate visualisation tools for insights and reporting.

#### **System Scalability**

* Compatible with scalable frameworks (e.g., PySpark).
* Maintained by an open-source community ensuring flexibility and adaptability.

#### **Performance Metrics**

* Profiling speed and accuracy.
* Scalability for large datasets.
* User feedback on visualisation and analysis outputs.