**Data Optimization Microservice**

A FastAPI microservice to clean, mask, and optimize textual data using NLP and Q-learning.

**🚀 Project Overview**

This project implements a **data optimization workflow** for text data, combining:

* **Data preprocessing** – handle missing values, assign default ratings.
* **Metadata extraction** – extract entities using NLP (spaCy).
* **Privacy masking** – anonymize sensitive information (names).
* **AI refinement** – sentiment analysis using Hugging Face Transformers + Q-learning to refine quality scores.
* **Storage simulation** – save outputs locally or optionally to Azure Blob.
* **API interface** – endpoints to optimize and retrieve refined data.

**🎯 Purpose & Uses**

**Why This Project Exists**

* **Automate Data Cleaning & Optimization**: Handles missing or inconsistent data and standardizes it for ML or analytics.
* **Prototype a Scalable ML Pipeline**: Demonstrates processing and refining data before feeding it into machine learning models.
* **Experiment with Q-Learning for Data Quality**: Uses reinforcement learning to iteratively refine numeric quality scores.
* **Provide API-Driven Access**: Users or services can POST new data and GET refined results through an API.
* **Demo Data Privacy & Masking Techniques**: Automatically masks personal names ([MASKED]) to protect sensitive information.

**Uses of This Project**

* Data preprocessing for ML and analytics.
* Sentiment analysis on text data.
* Refinement of data quality scores using reinforcement learning.
* API-driven workflow for real-time data optimization.
* Simulated cloud storage for processed data.

**🧰 Technologies & ML Techniques**

* **Python 3.10+**
* **FastAPI** – API framework
* **Uvicorn** – ASGI server
* **Pandas** – data handling
* **spaCy** – NLP entity extraction
* **Transformers (Hugging Face)** – sentiment analysis
* **PyTorch** – backend for Transformers
* **Q-Learning** – reinforcement learning for refining quality scores
* **Regex** – mask sensitive information
* **Azure Storage SDK** (optional) – simulate external storage

**🗂 Project Structure**

data\_optimizer\_microservice/

├── app.py # FastAPI main application

├── data\_optimizer/

│ ├── \_\_init\_\_.py

│ ├── data\_optimizer.py # Main data processing pipeline

│ ├── qlearning.py # Q-learning agent and helpers

│ ├── storage\_simulator.py # Azure/local storage simulation

│ └── utils.py # Rate limiter, masking functions

├── input\_data.json # Sample input for testing

├── outputs/ # Pipeline output folder (created at runtime)

│ ├── cleaned\_data.json

│ ├── metadata.json

│ ├── refined\_data.json

│ ├── log.json

│ └── q\_table.json

├── requirements.txt # Python dependencies

├── .env # Optional: Azure credentials

└── README.md # Project documentation

**⚡ How to Run**

1. **Clone the repository**

git clone https://github.com/Gokul099/-Data\_Optimization\_Microservice-.git

cd data\_optimizer\_microservice

1. **Create a virtual environment & install dependencies**

python -m venv .venv

source .venv/bin/activate # Linux/Mac

.venv\Scripts\activate # Windows

pip install -r requirements.txt

python -m spacy download en\_core\_web\_sm

1. **Run FastAPI app**

uvicorn app:app --reload

1. **Access API**

* **Optimize data** (POST):

POST http://127.0.0.1:8000/optimize

Content-Type: application/json

Body: input\_data.json

* **Retrieve refined data** (GET with API Key):

GET http://127.0.0.1:8000/retrieve

Header: X-API-Key: secret123

**📦 Sample Input (input\_data.json)**

[

{"text": "John from HR rated the document 7/10 on 2025-08-22T14:00:00Z", "rating": 7, "timestamp": "2025-08-22T14:00:00Z"},

{"text": "Sarah in Marketing loved the report. Rating: 9/10", "rating": 9, "timestamp": "2025-08-22T15:00:00Z"},

{"text": "The guide by Tom was confusing. No rating.", "rating": null, "timestamp": "2025-08-22T16:00:00Z"},

{"text": "Excellent project update. Rating: 8/10", "rating": 8, "timestamp": null},

{"text": "Content by Jane was error-free. Rating: 10/10", "rating": 10, "timestamp": "2025-08-22T18:00:00Z"}

]

**🏁 Sample Output (refined\_data.json)**

[

{

"asset\_id": "asset\_1a2b3c4d",

"text": "[MASKED] from HR rated the document 7/10 on 2025-08-22T14:00:00Z",

"orig\_rating": 7,

"refined\_quality": 7.1,

"sentiment": "POSITIVE",

"confidence": 0.99,

"timestamp": "2025-08-22T14:00:00Z"

},

{

"asset\_id": "asset\_5e6f7g8h",

"text": "[MASKED] in Marketing loved the report. Rating: 9/10",

"orig\_rating": 9,

"refined\_quality": 9.0,

"sentiment": "POSITIVE",

"confidence": 0.98,

"timestamp": "2025-08-22T15:00:00Z"

}

]

**✅ Notes**

* Make sure .venv and other large files are **not pushed to GitHub**.
* Use **Postman or curl** to test the endpoints.
* The Q-learning component refines quality scores **iteratively** based on sentiment.