# **Othor AI – Take-Home Assignment**

**Role**: Full Stack AI Developer

**Level**: 1–2 Years Experience

**Duration**: ~8–12 hours

**Submission**: Public GitHub Repository with README and Video recording

## **Overview**

At Othor AI, we're building intelligent platforms that help users derive insights from data instantly. As part of the hiring process, this assignment will evaluate your ability to work with real-world AI backend and frontend challenges.

Your task is to build a **Mini AI Analyst as a Service (AaaS)** — a microservice that allows users to upload business-related CSVs, automatically analyze the data, train a machine learning model if applicable, and return insights and predictions via an API and dashboard.

This assignment will assess your understanding of backend development, model integration, system design, and full-stack thinking.

## **Objectives**

### **Part 1: CSV Ingestion and Metadata Engine**

Build a FastAPI backend that exposes an /upload endpoint which:

* Accepts .csv files up to 50MB
* Streams the file (avoid full memory load)
* Infers schema:
  + Column types (categorical, numerical, datetime, boolean)
  + Unique value counts
  + Null percentage
  + Flags for high cardinality and constant columns

Implement a /profile endpoint to:

* Return a metadata report for the uploaded file:
  + Outliers
  + Skewness
  + Pairwise correlations
  + Imbalanced columns
  + Data leakage detection (e.g., if a feature is highly correlated with the target)

Each uploaded file should be linked to a session token (UUID) for further API access.

### **Part 2: AutoML Model Pipeline**

Create a /train endpoint that:

* Accepts a column name as the target (optional: infer it if labeled as target, label, etc.)
* Performs preprocessing:
  + Encode categorical columns
  + Handle missing data
* Trains a basic classification or regression model:
  + Recommended: RandomForest, Logistic Regression, XGBoost
* Returns:
  + Evaluation metrics (accuracy, precision, recall, F1-score or RMSE, R²)
  + Model file path or access token
  + Feature importances or SHAP values (optional)
* Save the model to disk and allow re-use

### **Part 3: Inference and Insight Generation**

Develop the following endpoints:

* /predict
  + Accepts new data and a model token
  + Returns predictions and confidence scores
* /summary
  + Returns a natural-language summary of the data and model:
    - Number of rows/columns
    - Target feature correlation
    - Top predictors
    - Model performance

(Optional but encouraged: use a local LLM or transformer model to generate human-readable summaries)

### **Part 4: Frontend Dashboard**

Develop a minimal UI using React/Next.js or plain HTML+JavaScript that allows:

* CSV upload
* Viewing parsed schema and profiling insights
* Triggering model training
* Viewing prediction results in table format
* Visualizing results (basic charts preferred)

The UI should handle loading states and server errors gracefully.

## **Bonus Features (Optional but Encouraged)**

* Implement background job processing for model training using Celery or similar
* Store uploaded files and model metadata in PostgreSQL or MongoDB
* Add user authentication with JWT (admin vs viewer access)
* Use an S3-compatible bucket for storage
* Include visual feature importance charts or clustering analysis
* Add a retry mechanism in inference endpoints for fault tolerance

## **Technical Requirements**

* **Python 3.9+**
* **FastAPI**
* **Scikit-learn or XGBoost**
* **Pandas, NumPy**
* **React/Next.js** (or plain JS if preferred)
* **Dockerized setup**
* Write at least basic unit tests for key endpoints

## **Sample Use Case**

A business user uploads a product-sales dataset.

Your system:

* Profiles the data
* Detects the schema and target variable (Churn)
* Trains a classification model
* Provides a summary and serves predictions via an API and dashboard interface

## **Evaluation Criteria**

| **Area** | **Assessment Focus** |
| --- | --- |
| Backend API Design | Modular FastAPI implementation, best practices |
| Machine Learning Logic | Sensible pipelines, model persistence |
| Data Profiling | Insightful statistics, smart detection logic |
| Summarization | Coherent and human-readable output |
| Frontend Integration | Functionality, state handling, responsiveness |
| Code Quality | Structure, readability, naming, error handling |
| Deployment | Functional Docker setup and environment config |
| Bonus Implementation | Innovation, architecture, optional features |

## **Submission Guidelines**

1. Upload your code to a **public GitHub repository**
2. Include a README.md with:
   * Setup instructions
   * How to run/test the app (locally or via Docker)
   * API documentation (basic usage examples)
   * Sample CSVs (if any)
   * Assumptions and limitations
   * Time taken and optional features attempted
3. Record a video as well explaining your approach