🧠 Machine Learning Engineering Challenge – Time Series Anomaly Detection

📄 Case Description

Build a system to perform **anomaly detection** on univariate time series data. The system should support training and inference for multiple distinct time series, each identified by a unique `series_id`. It must persist model weights, support versioning, and provide real-time predictions.

Concepts

Univariate Time Series

A univarite time series can be described by the following object

```
from typing import Sequence

from pydantic import BaseModel, Field

class DataPoint(BaseModel):
    timestamp: int = Field(..., description =" Unix timestamp of the time the data point was
collected")
    value: float = Field(...,  description = "Value of the time series measured at time
`timestamp`")

class TimeSeries(BaseModel):
    data: Sequence[DataPoint] = Field(..., description = "List of datapoints, ordered in time, of
subsequent measurements of some quantity")
```

Time series area realy useful for several group of applications, like Weather Prediction, Econometrics, etc.

Anomaly Detection

Usually, time series data is used for forecasting purposes. In this scenario, however we are more interested in deciding if a point belongs to the so-called `learned distribution`. This process is commonly refered to as `Anomaly Detection` or `Outlier Detection`. To know more about it and some techniques, go to Scikit-learn's Documentation.

The model performance is not the scope of this challenge, so we are providing you with a basic model for Anomaly Detection that you can use as your model of choice.

```
from pydatinc import BaseModel
import numpy as np

class AnomalyDetectionModel:
    def fit(self, data: TimeSeries) -> "AnomalyDetection":
        values_stream = [d.value for d in data]
        self.mean = np.mean(values_stream)
        self.std = np.std(values_stream)

    def predict(self, data_point: DataPoint) -> bool:
        return data_point.value > self.mean + 3 * self.std
```

🚀 Features

- Train anomaly detection models from timestamped values via API
- Maintain separate models per `series_id` with versioning support
- Persist trained models for reuse
- Predict if new points are anomalous given a `series_id`
- Return model version and prediction
- Report system-level performance metrics (latency, load)

🧪 API Specification

See the OpenAPI file sent along this document. To a better visualization copy and paste the file content into the swagger editor.

💡 Optional Enhancements

- **Performance Testing**: Include benchmarking results under load (e.g. 100 parallel inferences)
- **Preflight Validation**: Reject training data that is insufficient, constant, or invalid
- **Visualization Tool**: Provide a `/plot?series_id=sensor_XYZ&version=v3` endpoint to show training data
- **Model Versioning**: Support retraining of the same `series_id`, versioning each model

📦 Deliverables

- GitHub repository with full implementation
- README with setup instructions and usage examples
- `.env.example` and any required secrets/configs
- Sample training and prediction requests
- If enhancements are implemented, include visualizations or benchmark results

✅ Evaluation Criteria

Criteria	Description
Functionality	Trains and predicts across multiple `series_id`s with persistence
Input Handling	Accepts raw JSON time series input, validates structure and values
Code Quality	Modular, readable, testable
API Design	RESTful, simple, and ergonomic
Scalability	Handles concurrent training and prediction requests
Performance Reporting	Demonstrates awareness of latency, throughput, and model usage

Good luck 🚀