**Unit Testing Description: testing.py**

This document thoroughly explains the unit testing code in 'testing.py'. The code tests various components of a sensor anomaly detection system, including data retrieval from a database and forecasting model predictions. The system uses a traffic light-based status indicator (Green, Amber, Red) to classify sensor readings as normal, warning, or anomalous. The testing framework relies on pytest and the unittest.mock library.

**🧩 1. Imports & Setup**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **What it does**: Brings in all external libraries needed.
* **Why it matters**: Enables test definition (pytest), mocking (unittest.mock), and real data manipulation (pandas, joblib, etc).

**🧩 2. Pytest Fixtures**

A screenshot of a computer program

AI-generated content may be incorrect. ...

* **Purpose**: Prepares reusable mock objects (cursor, DB connection, Prophet model).
* **Used in**: Other test cases that simulate database and model behavior.
* **Why**: Prevents needing a real MySQL server or trained ML model.

**🧩 3. Testing DB Connection:**

**A close-up of a white background

AI-generated content may be incorrect.**

* **What it does**: Mocks mysql.connector.connect, simulates connecting to DB.
* **Checks**: Whether the connection is made and print('Connected to database') is called.
* **Technique**: Uses exec() to simulate code inside a module that connects globally.

**🧩 4. Get Latest Data for Line 4:**

**A close-up of a text

AI-generated content may be incorrect.**

* **Simulates**: A DB row returned from line4\_sensors.
* **Checks**: If SQL is built correctly and result is parsed into a dictionary.
* **Example Result**:

json

Copy code

{"id": 1, "timestamp": ..., "r01": 100, ..., "r08": 135}

**🧩 5. Get Latest Data for Line 5**

**A close-up of a computer code

AI-generated content may be incorrect.**

* Similar to Line 4 test but handles 17 sensors.
* **Extra Check**: Makes sure r17 (last sensor) is parsed correctly.

**🧩 6. Green Status Test (Normal Reading):**

**A blue text on a white background

AI-generated content may be incorrect.**

* **Scenario**: Sensor value is inside the predicted range (120 between 110–130).
* **Result**: Prints "🟢 GREEN (Normal)".

**🧩 7. Amber Status Test – Lower:**

**A close-up of a computer code

AI-generated content may be incorrect.**

* **Scenario**: Value 100, slightly below 110.
* **Result**: "🟠 AMBER (Warning)" due to lower range breach.

**🧩 8. Amber Status Test – Upper:**

**A close-up of a computer code

AI-generated content may be incorrect.**

* **Scenario**: Value 135, slightly above 130.
* **Checks**: Classification into upper amber zone (130 to 145).

**🧩 9. Red Status Test (Outlier):**

**A close-up of a computer code

AI-generated content may be incorrect.**

* **Scenario**: Value = 150, well beyond upper amber bound (145).
* **Result**: "🔴 RED (Anomaly)"

**🧩 10. Test the Main Loop Logic:**

**A close-up of a smiley

AI-generated content may be incorrect.**

* **Mocks**:
  + get\_latest\_data returns fixed readings.
  + check\_anomalies is a mock.
* **Loop**: Simulates one cycle of the real monitoring system.
* **Checks**: All sensors are processed, and sleep is called once.

**🧩 11. Integration Test (End-to-End Simulation):**

**A close-up of a text

AI-generated content may be incorrect.**

* **Purpose**: Combines:
  + Real check\_anomalies logic
  + Real get\_latest\_data
  + Mocked database and forecast model
* **What’s Tested**:
  + Each sensor is forecasted and classified.
  + Output is printed.
  + Sleep call confirms timed loop.

# 💡 Summary

The tests are structured to confirm each component of the system functions both individually and together. Mocking allows fast and safe testing without requiring real database connections or model files. This makes the test suite ideal for continuous integration pipelines, regression testing, and development validation.