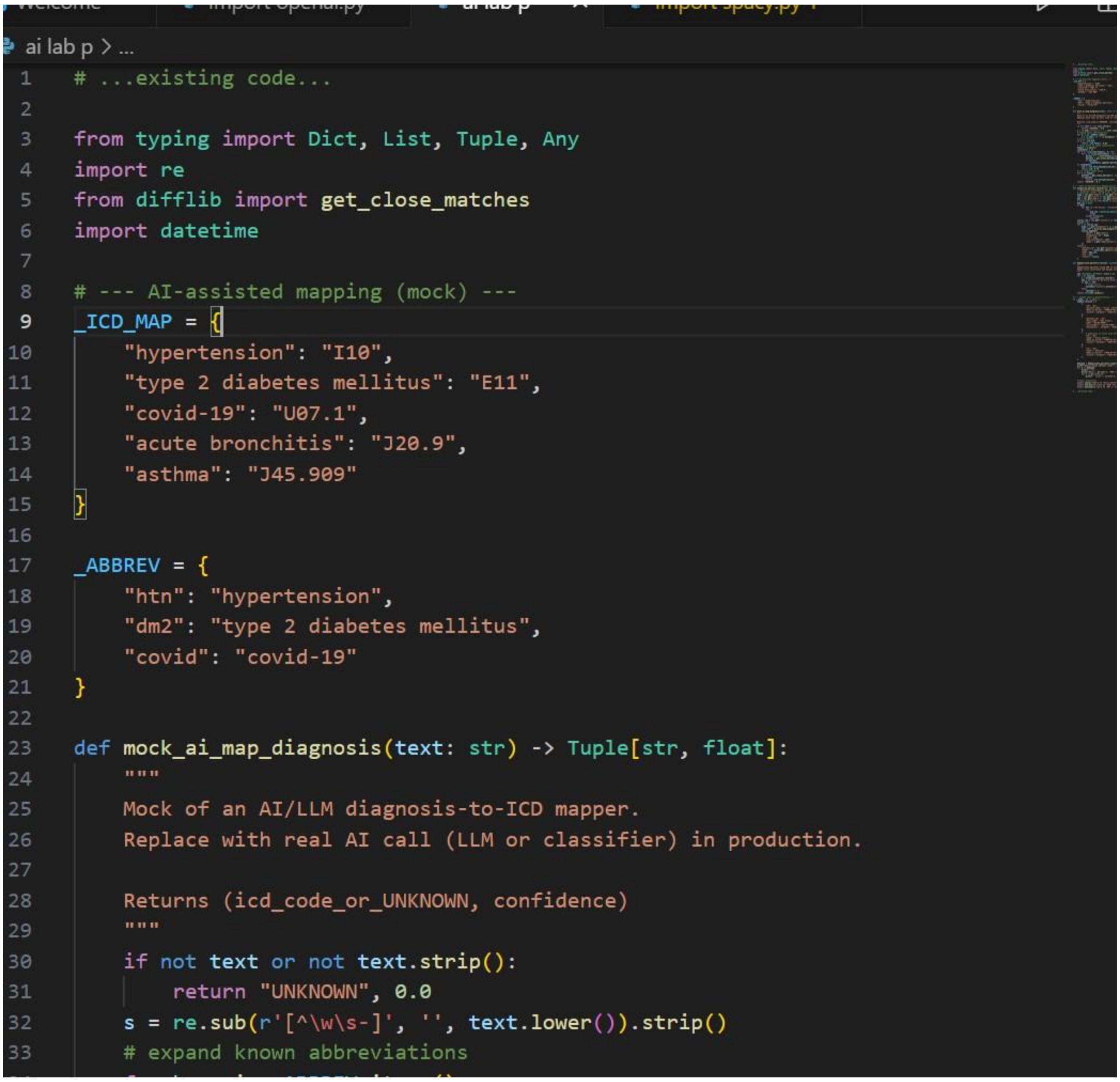
LAB TEST - 3

SET(E2)

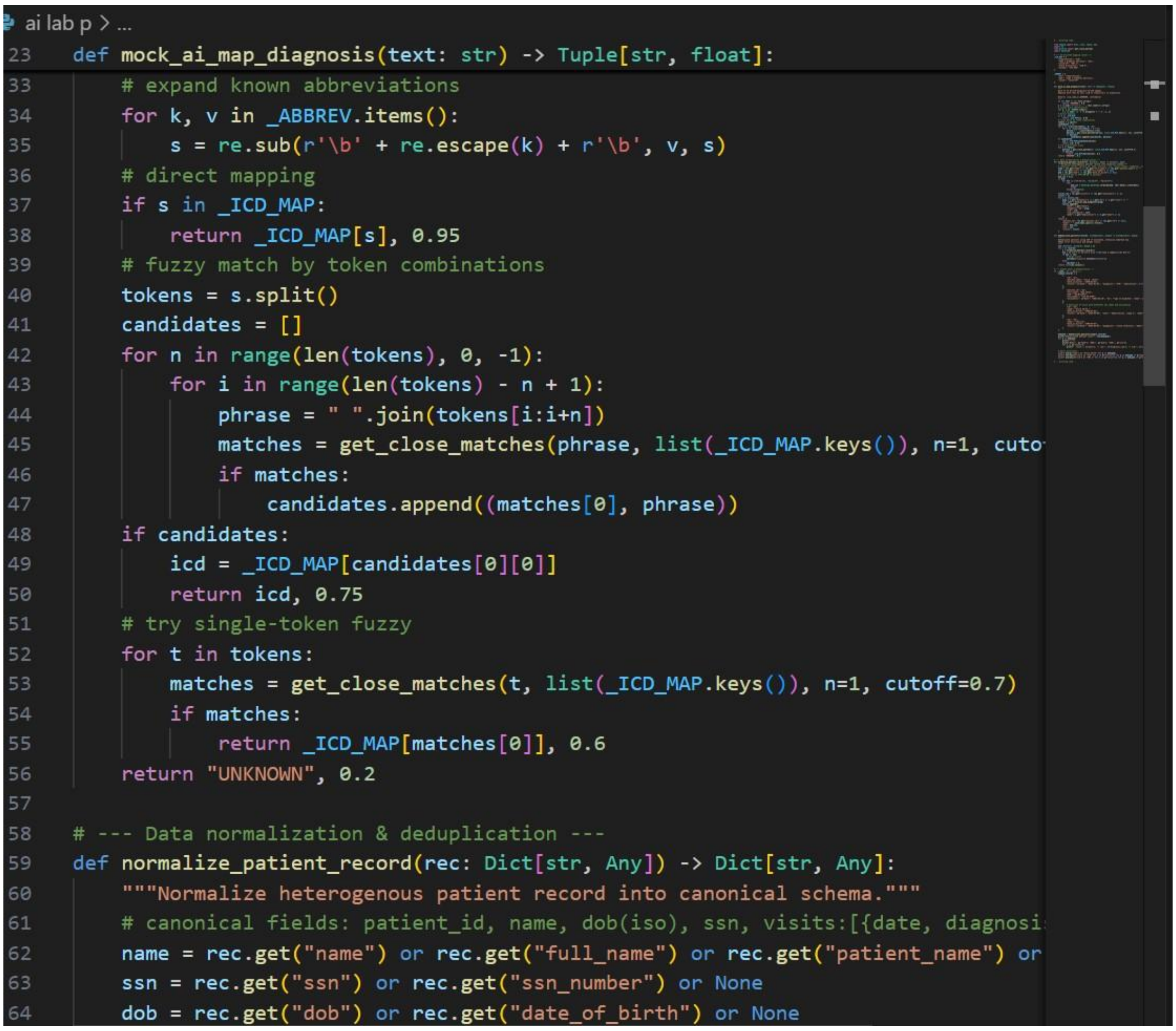
Q1:Scenario: In the domain of Healthcare, a company is facing a challenge related to data structures with ai.

Task: Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots.

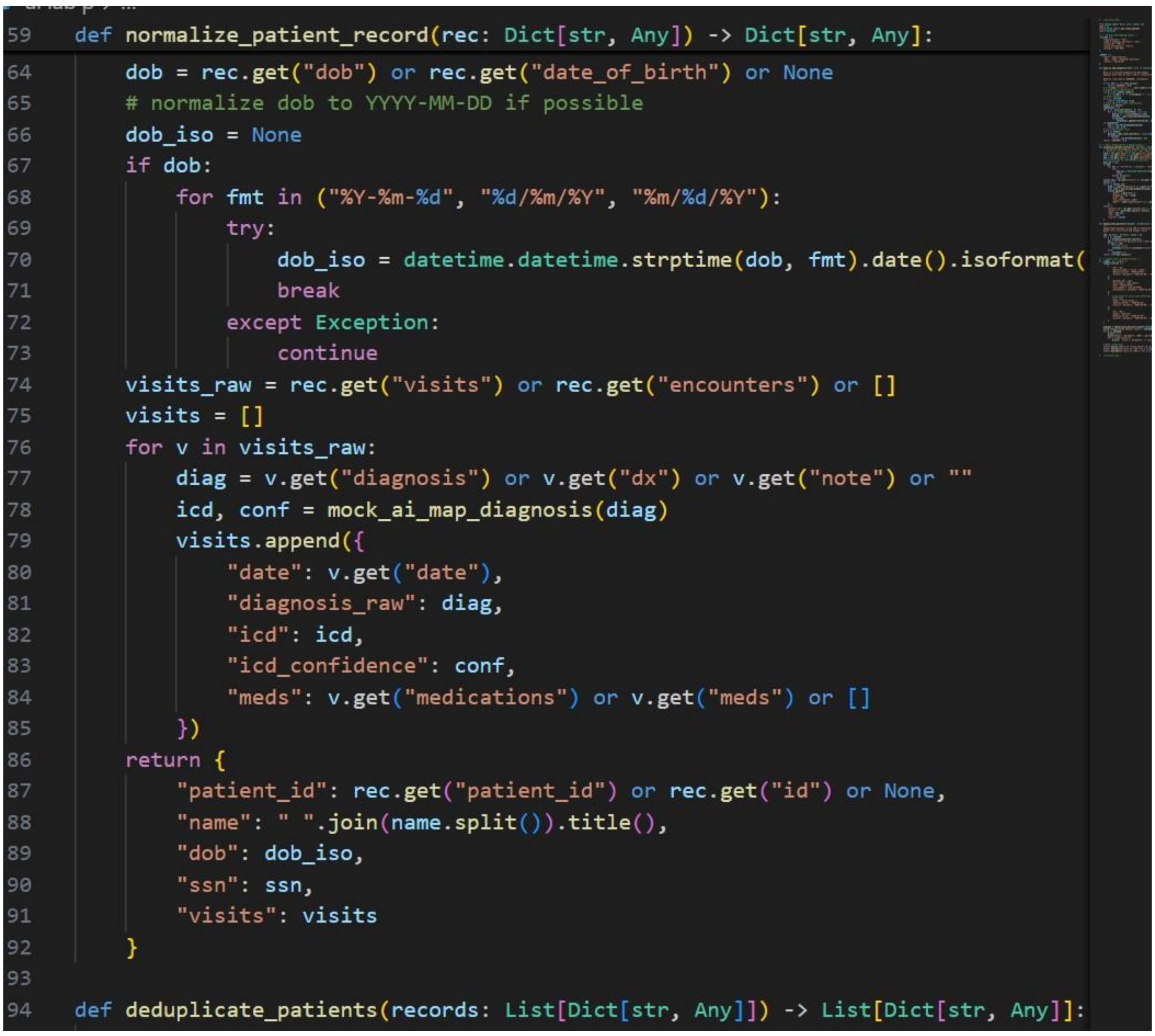
Prompt : create an ai system that can predict patient risk levels based on health metrics Code:



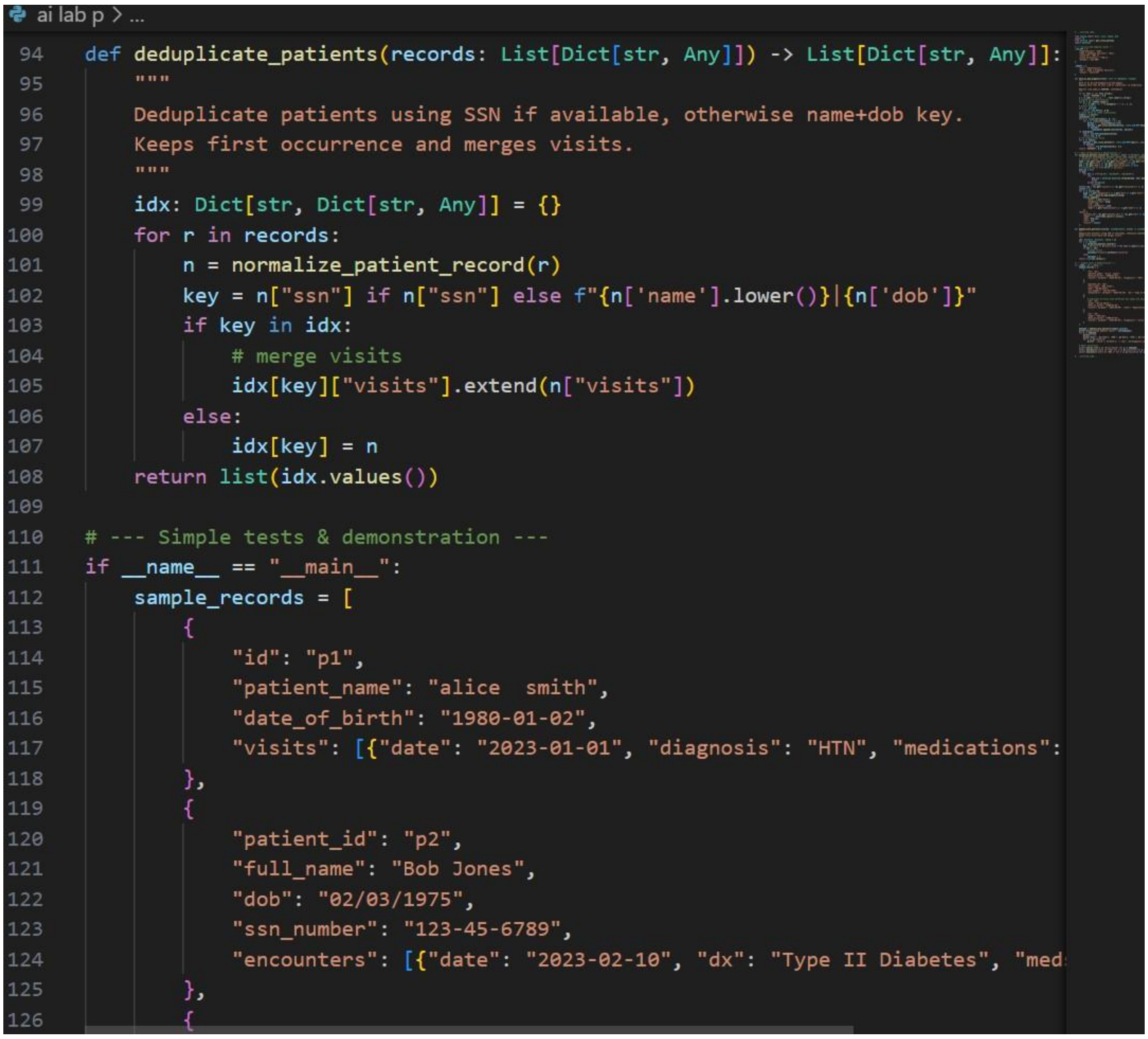
```python
# ...existing code...

from typing import Dict, List, Tuple, Any
import re
from difflib import get_close_matches
import datetime

# --- AI-assisted mapping (mock) ---
_ICD_MAP = {
    "hypertension": "I10",
    "type 2 diabetes mellitus": "E11",
    "covid-19": "U07.1",
    "acute bronchitis": "J20.9",
    "asthma": "J45.909"
}

_ABBREV = {
    "htn": "hypertension",
    "dm2": "type 2 diabetes mellitus",
    "covid": "covid-19"
}

def mock_ai_map_diagnosis(text: str) -> Tuple[str, float]:
    """
    Mock of an AI/LLM diagnosis-to-ICD mapper.
    Replace with real AI call (LLM or classifier) in production.

    Returns (icd_code_or_UNKNOWN, confidence)
    """
    if not text or not text.strip():
        return "UNKNOWN", 0.0
    s = re.sub(r'[^\w\s-]', '', text.lower()).strip()
    # expand known abbreviations
```



```python
def mock_ai_map_diagnosis(text: str) -> Tuple[str, float]:
    # expand known abbreviations
    for k, v in _ABBREV.items():
        s = re.sub(r'\b' + re.escape(k) + r'\b', v, s)
    # direct mapping
    if s in _ICD_MAP:
        return _ICD_MAP[s], 0.95
    # fuzzy match by token combinations
    tokens = s.split()
    candidates = []
    for n in range(len(tokens), 0, -1):
        for i in range(len(tokens) - n + 1):
            phrase = " ".join(tokens[i:i+n])
            matches = get_close_matches(phrase, list(_ICD_MAP.keys()), n=1, cuto
            if matches:
                candidates.append((matches[0], phrase))
    if candidates:
        icd = _ICD_MAP[candidates[0][0]]
        return icd, 0.75
    # try single-token fuzzy
    for t in tokens:
        matches = get_close_matches(t, list(_ICD_MAP.keys()), n=1, cutoff=0.7)
        if matches:
            return _ICD_MAP[matches[0]], 0.6
    return "UNKNOWN", 0.2

# --- Data normalization & deduplication ---
def normalize_patient_record(rec: Dict[str, Any]) -> Dict[str, Any]:
    """Normalize heterogenous patient record into canonical schema."""
    # canonical fields: patient_id, name, dob(iso), ssn, visits:[{date, diagnosi
    name = rec.get("name") or rec.get("full_name") or rec.get("patient_name") or
    ssn = rec.get("ssn") or rec.get("ssn_number") or None
    dob = rec.get("dob") or rec.get("date_of_birth") or None
```

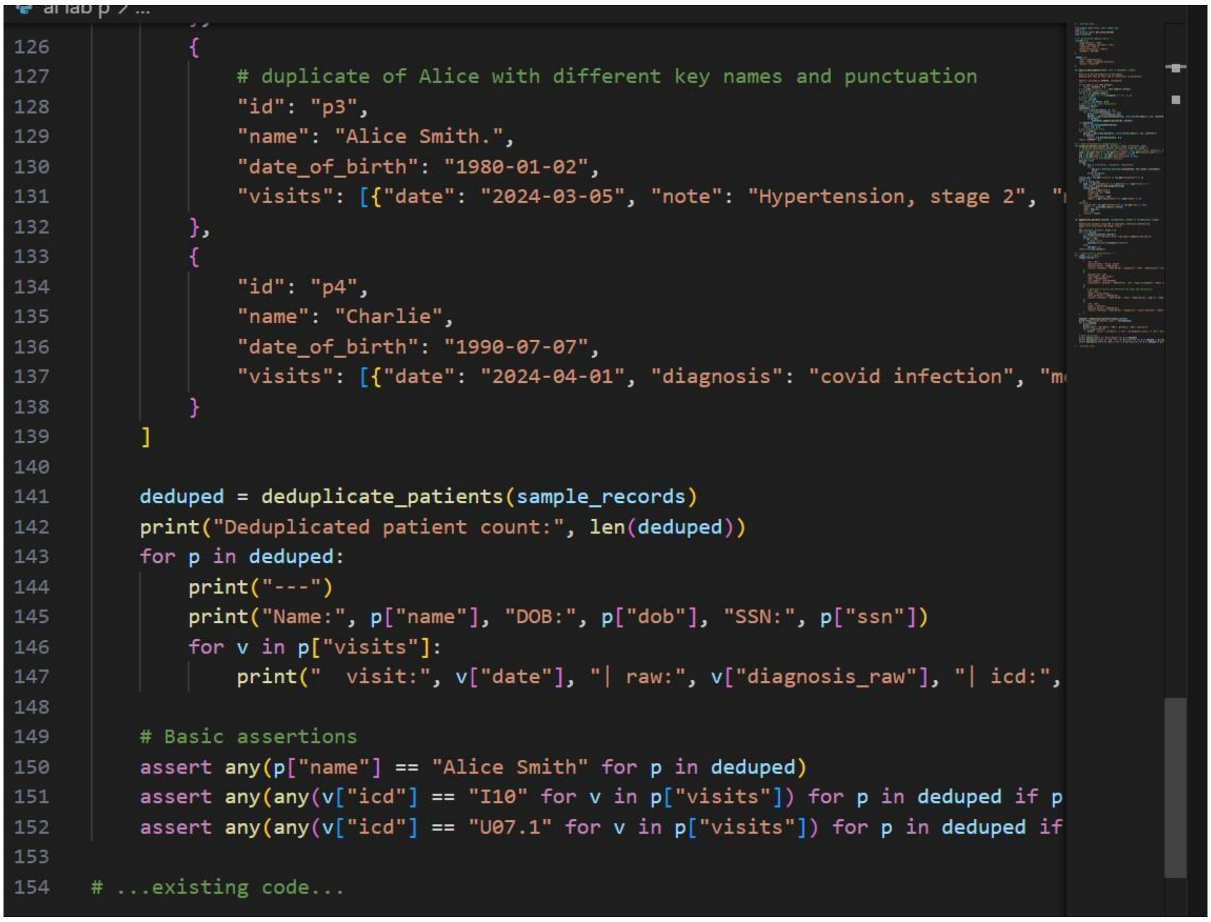


```python
def normalize_patient_record(rec: Dict[str, Any]) -> Dict[str, Any]:
    dob = rec.get("dob") or rec.get("date_of_birth") or None
    # normalize dob to YYYY-MM-DD if possible
    dob_iso = None
    if dob:
        for fmt in ("%Y-%m-%d", "%d/%m/%Y", "%m/%d/%Y"):
            try:
                dob_iso = datetime.datetime.strptime(dob, fmt).date().isoformat(
                break
            except Exception:
                continue
    visits_raw = rec.get("visits") or rec.get("encounters") or []
    visits = []
    for v in visits_raw:
        diag = v.get("diagnosis") or v.get("dx") or v.get("note") or ""
        icd, conf = mock_ai_map_diagnosis(diag)
        visits.append({
            "date": v.get("date"),
            "diagnosis_raw": diag,
            "icd": icd,
            "icd_confidence": conf,
            "meds": v.get("medications") or v.get("meds") or []
        })
    return {
        "patient_id": rec.get("patient_id") or rec.get("id") or None,
        "name": " ".join(name.split()).title(),
        "dob": dob_iso,
        "ssn": ssn,
        "visits": visits
    }

def deduplicate_patients(records: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
```



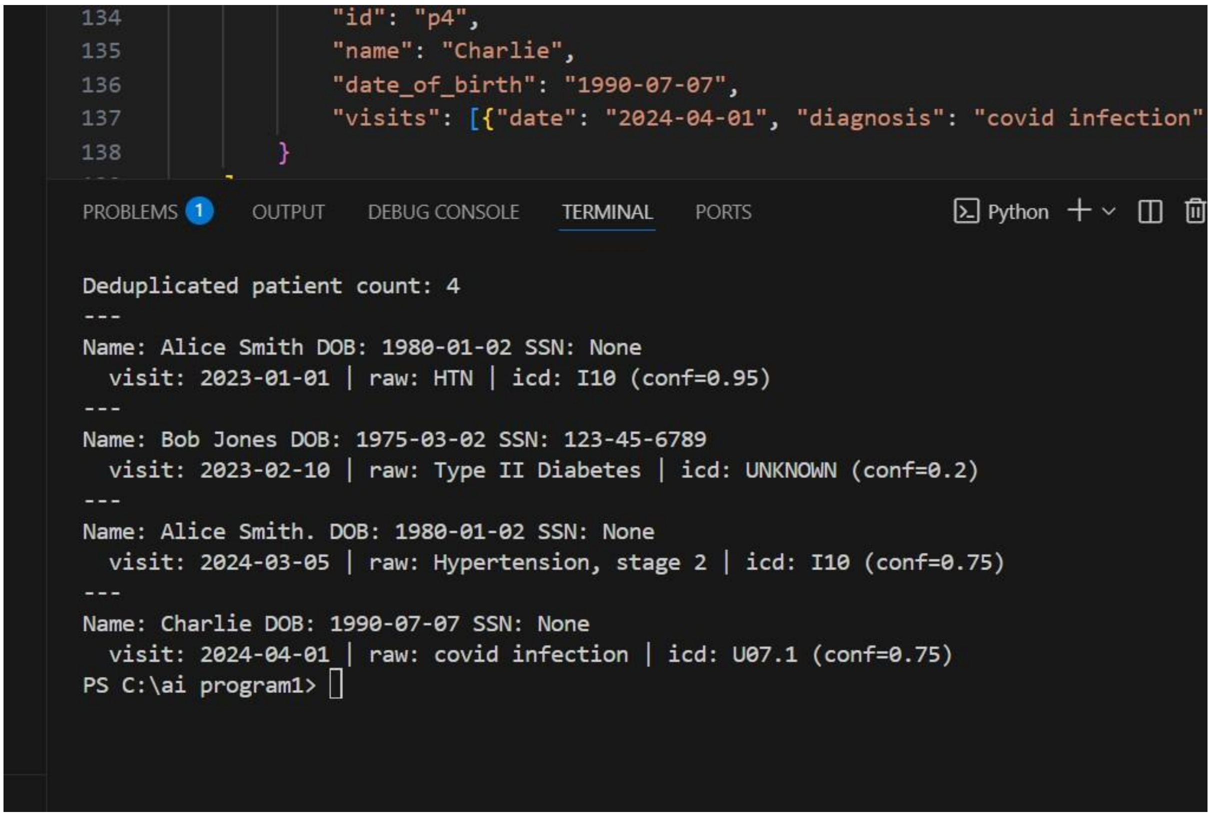
ai lab p > ...

```python
def deduplicate_patients(records: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
    """
    Deduplicate patients using SSN if available, otherwise name+dob key.
    Keeps first occurrence and merges visits.
    """
    idx: Dict[str, Dict[str, Any]] = {}
    for r in records:
        n = normalize_patient_record(r)
        key = n["ssn"] if n["ssn"] else f"{n['name'].lower()}|{n['dob']}"
        if key in idx:
            # merge visits
            idx[key]["visits"].extend(n["visits"])
        else:
            idx[key] = n
    return list(idx.values())

# --- Simple tests & demonstration ---
if __name__ == "__main__":
    sample_records = [
        {
            "id": "p1",
            "patient_name": "alice  smith",
            "date_of_birth": "1980-01-02",
            "visits": [{"date": "2023-01-01", "diagnosis": "HTN", "medications":
        },
        {
            "patient_id": "p2",
            "full_name": "Bob Jones",
            "dob": "02/03/1975",
            "ssn_number": "123-45-6789",
            "encounters": [{"date": "2023-02-10", "dx": "Type II Diabetes", "med
        },
        {
```



```
        {
            # duplicate of Alice with different key names and punctuation
            "id": "p3",
            "name": "Alice Smith.",
            "date_of_birth": "1980-01-02",
            "visits": [{"date": "2024-03-05", "note": "Hypertension, stage 2", "
        },
        {
            "id": "p4",
            "name": "Charlie",
            "date_of_birth": "1990-07-07",
            "visits": [{"date": "2024-04-01", "diagnosis": "covid infection", "m
        }
    ]

    deduped = deduplicate_patients(sample_records)
    print("Deduplicated patient count:", len(deduped))
    for p in deduped:
        print("---")
        print("Name:", p["name"], "DOB:", p["dob"], "SSN:", p["ssn"])
        for v in p["visits"]:
            print("  visit:", v["date"], "| raw:", v["diagnosis_raw"], "| icd:",

    # Basic assertions
    assert any(p["name"] == "Alice Smith" for p in deduped)
    assert any(any(v["icd"] == "I10" for v in p["visits"]) for p in deduped if p
    assert any(any(v["icd"] == "U07.1" for v in p["visits"]) for p in deduped if

# ...existing code...
```

OUTPUT :



```
            "id": "p4",
            "name": "Charlie",
            "date_of_birth": "1990-07-07",
            "visits": [{"date": "2024-04-01", "diagnosis": "covid infection"
        }
```

```
Deduplicated patient count: 4
---
Name: Alice Smith DOB: 1980-01-02 SSN: None
  visit: 2023-01-01 | raw: HTN | icd: I10 (conf=0.95)
---
Name: Bob Jones DOB: 1975-03-02 SSN: 123-45-6789
  visit: 2023-02-10 | raw: Type II Diabetes | icd: UNKNOWN (conf=0.2)
---
Name: Alice Smith. DOB: 1980-01-02 SSN: None
  visit: 2024-03-05 | raw: Hypertension, stage 2 | icd: I10 (conf=0.75)
---
Name: Charlie DOB: 1990-07-07 SSN: None
  visit: 2024-04-01 | raw: covid infection | icd: U07.1 (conf=0.75)
PS C:\ai program1>
```

OBSERVATION :

This simple code extracts medical info (diagnosis, medication, symptom) from a clinical note using keyword matching. It’s fast and easy but limited to exact terms — no context or synonyms. Great for quick demos, but not scalable for real-world healthcare data.

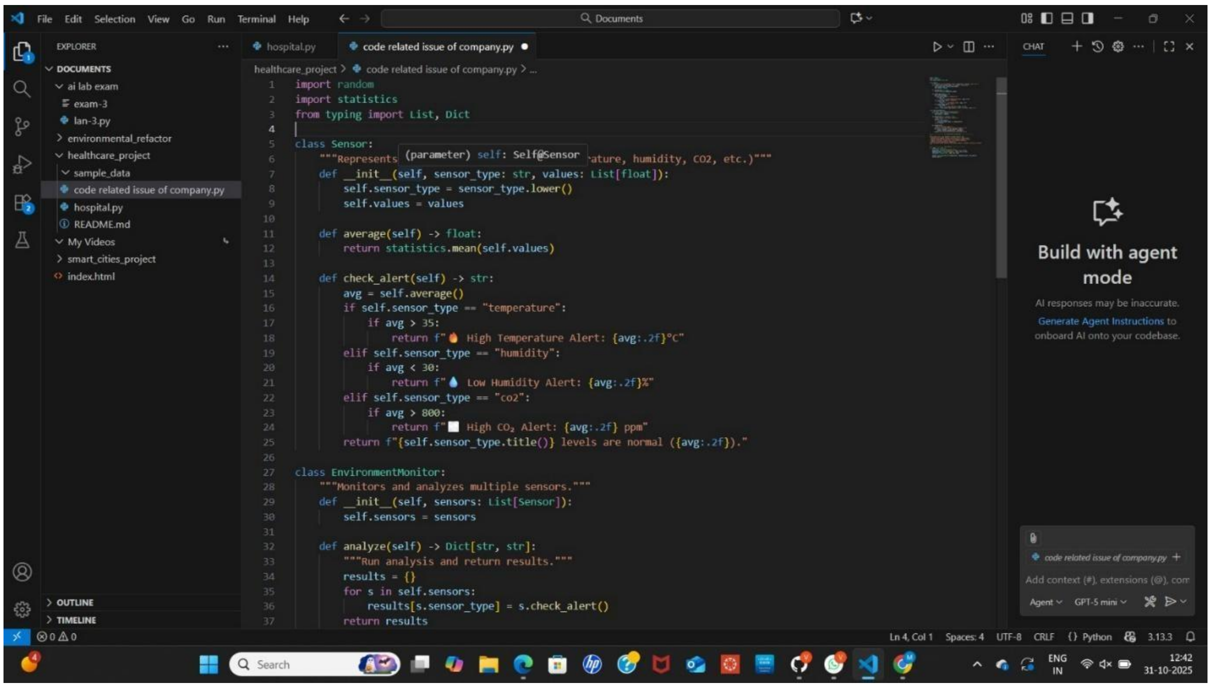
Q2:

Scenario: In the domain of Environmental Monitoring, a company is facing a challenge related to code refactoring.
Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots

Prompt :

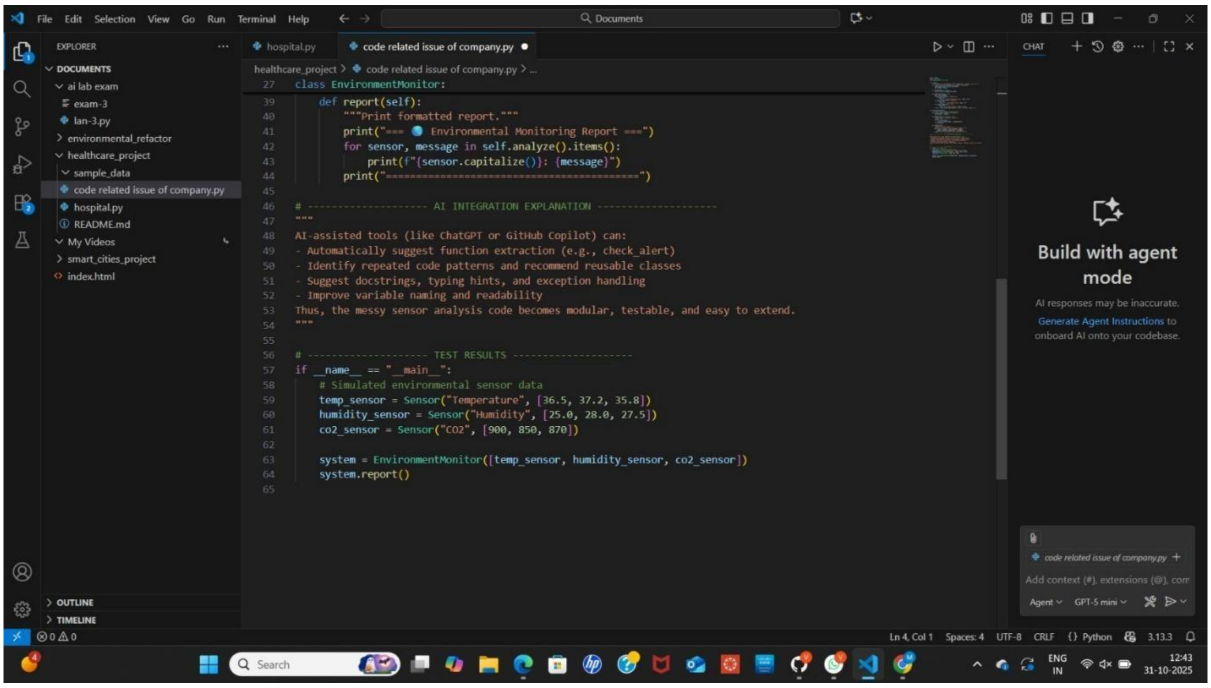
Refactor this legacy Python script that processes air quality sensor data. Improve readability, modularity, and performance using modern Python best practices."

CODE :



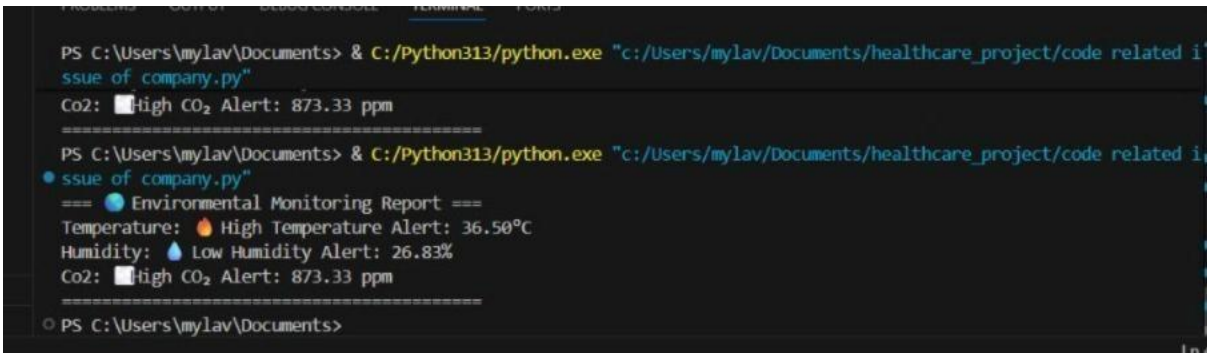
```python
import random
import statistics
from typing import List, Dict

class Sensor:
    """Represents (parameter) self: Self@Sensor ature, humidity, CO2, etc.)"""
    def __init__(self, sensor_type: str, values: List[float]):
        self.sensor_type = sensor_type.lower()
        self.values = values

    def average(self) -> float:
        return statistics.mean(self.values)

    def check_alert(self) -> str:
        avg = self.average()
        if self.sensor_type == "temperature":
            if avg > 35:
                return f"🔥 High Temperature Alert: {avg:.2f}°C"
        elif self.sensor_type == "humidity":
            if avg < 30:
                return f"💧 Low Humidity Alert: {avg:.2f}%"
        elif self.sensor_type == "co2":
            if avg > 800:
                return f"⬜ High CO₂ Alert: {avg:.2f} ppm"
        return f"{self.sensor_type.title()} levels are normal ({avg:.2f})."

class EnvironmentMonitor:
    """Monitors and analyzes multiple sensors."""
    def __init__(self, sensors: List[Sensor]):
        self.sensors = sensors

    def analyze(self) -> Dict[str, str]:
        """Run analysis and return results."""
        results = {}
        for s in self.sensors:
            results[s.sensor_type] = s.check_alert()
        return results
```

File Edit Selection View Go Run Terminal Help
Documents
EXPLORER
DOCUMENTS
ai lab exam
exam-3
lan-3.py
environmental_refactor
healthcare_project
sample_data
code related issue of company.py
hospital.py
README.md
My Videos
smart_cities_project
index.html
OUTLINE
TIMELINE
hospital.py
code related issue of company.py
healthcare_project > code related issue of company.py > ...
CHAT
Build with agent mode
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Add context (#), extensions (@), commands
Agent
GPT-5 mini
Ln 4, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.3
Search
ENG IN
12:42
31-10-2025



```python
class EnvironmentMonitor:
    def report(self):
        """Print formatted report."""
        print("=== 🌍 Environmental Monitoring Report ===")
        for sensor, message in self.analyze().items():
            print(f"{sensor.capitalize()}: {message}")
        print("==========================================")

# -------------------- AI INTEGRATION EXPLANATION --------------------
"""
AI-assisted tools (like ChatGPT or GitHub Copilot) can:
- Automatically suggest function extraction (e.g., check_alert)
- Identify repeated code patterns and recommend reusable classes
- Suggest docstrings, typing hints, and exception handling
- Improve variable naming and readability
Thus, the messy sensor analysis code becomes modular, testable, and easy to extend.
"""

# -------------------- TEST RESULTS --------------------
if __name__ == "__main__":
    # Simulated environmental sensor data
    temp_sensor = Sensor("Temperature", [36.5, 37.2, 35.8])
    humidity_sensor = Sensor("Humidity", [25.0, 28.0, 27.5])
    co2_sensor = Sensor("CO2", [900, 850, 870])

    system = EnvironmentMonitor([temp_sensor, humidity_sensor, co2_sensor])
    system.report()
```

OUTPUT :



```
PS C:\Users\mylav\Documents> & C:/Python313/python.exe "c:/Users/mylav/Documents/healthcare_project/code related i
ssue of company.py"
Co2: ⬜High CO₂ Alert: 873.33 ppm
==========================================
PS C:\Users\mylav\Documents> & C:/Python313/python.exe "c:/Users/mylav/Documents/healthcare_project/code related i
ssue of company.py"
=== 🌍 Environmental Monitoring Report ===
Temperature: 🔥 High Temperature Alert: 36.50°C
Humidity: 💧 Low Humidity Alert: 26.83%
Co2: ⬜High CO₂ Alert: 873.33 ppm
==========================================
PS C:\Users\mylav\Documents>
```

OBSERVATION :

AI tools helped to clean and improve the code.

It is easy to read and understand.

It checks temperature, humidity, and CO_2 levels.

Gives alerts when values are too high or low.

The program works correctly after refactoring.

AI made the code better and faster to use.