Lab Test-03

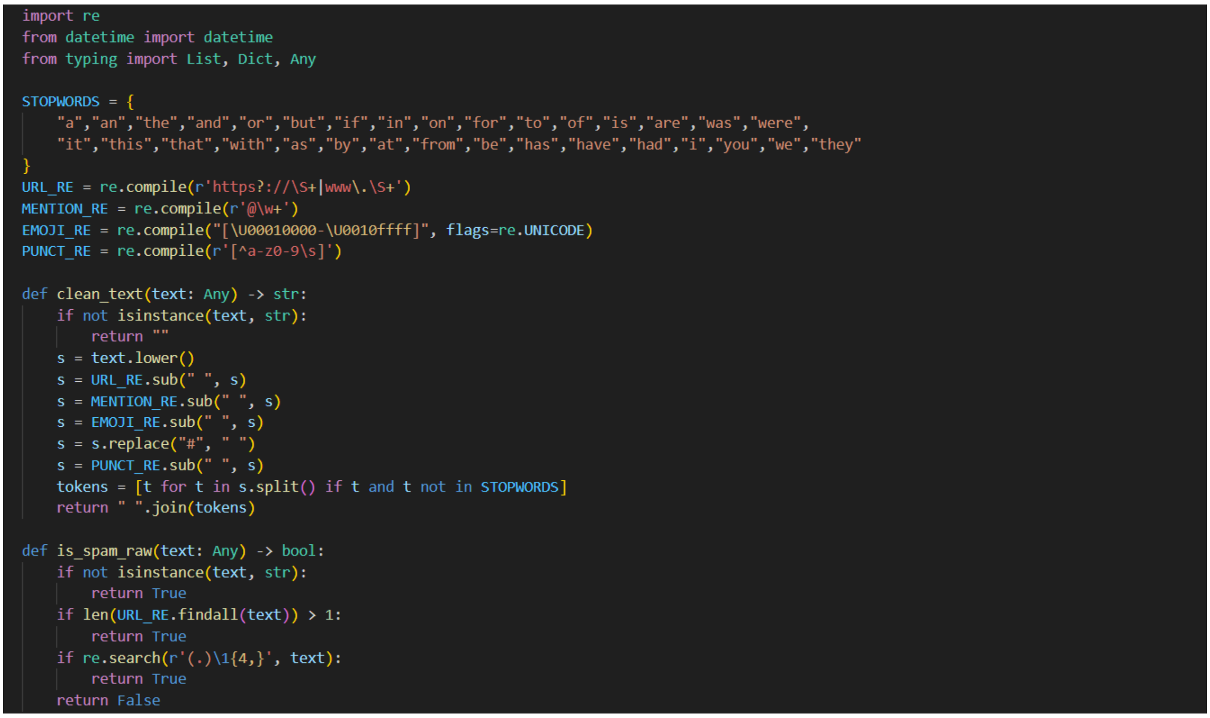
Name: D. Sravika Reddy Roll No: 2403a510d0

Batch: 05 Course: Ai Assisted Coding

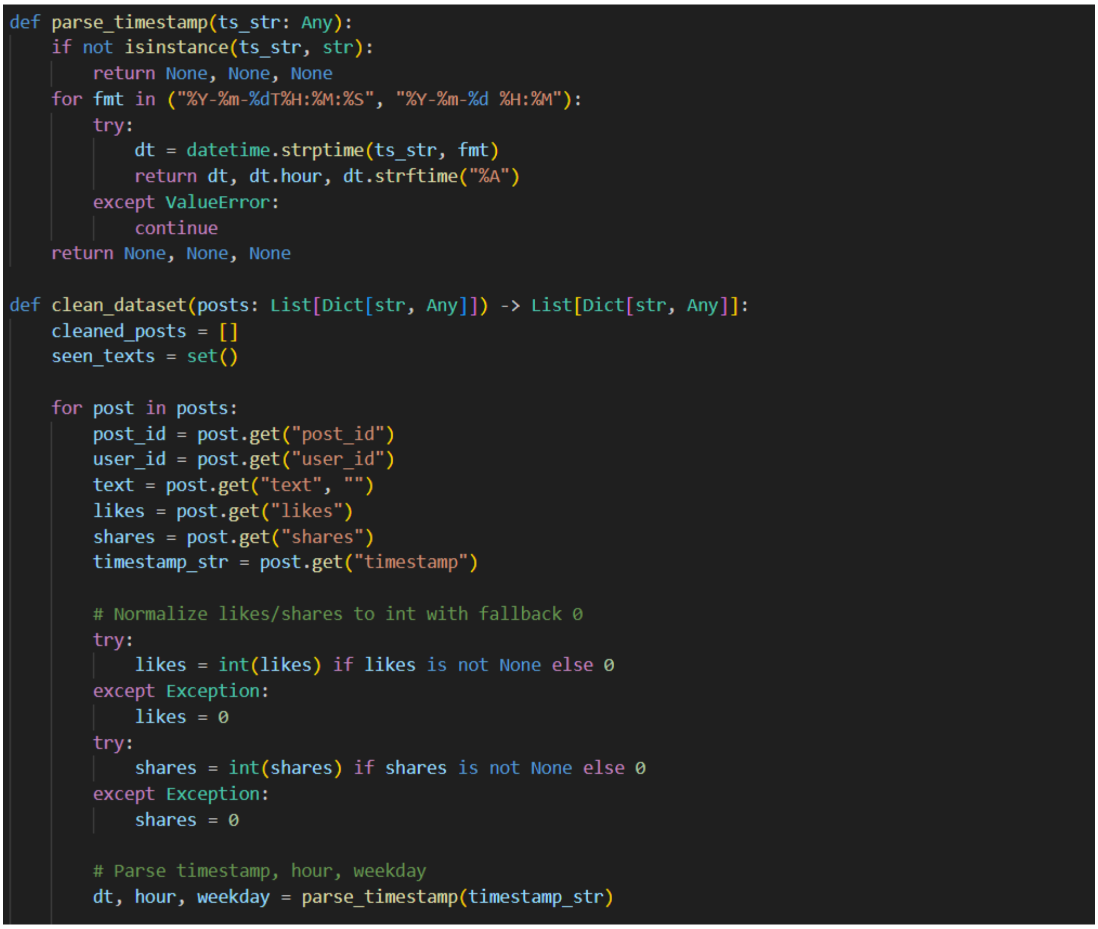
Set E4
Q1:
Scenario: In the Education sector, a company faces a challenge related to code refactoring.
Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

Prompt: Write a Python module that cleans social media posts. Requirements: remove URLs/mentions/emojis/punctuation, lowercase and remove stopwords, fill missing likes/shares with 0, parse timestamp and extract hour and weekday, flag/remove spam (>1 URL, long repeated chars, very short cleaned text), drop duplicate cleaned posts (keep first), and return DataFrame with columns: post_id, user_id, clean_text, likes, shares, timestamp, hour, weekday. Include a small sample and basic assertions when run as __main__.

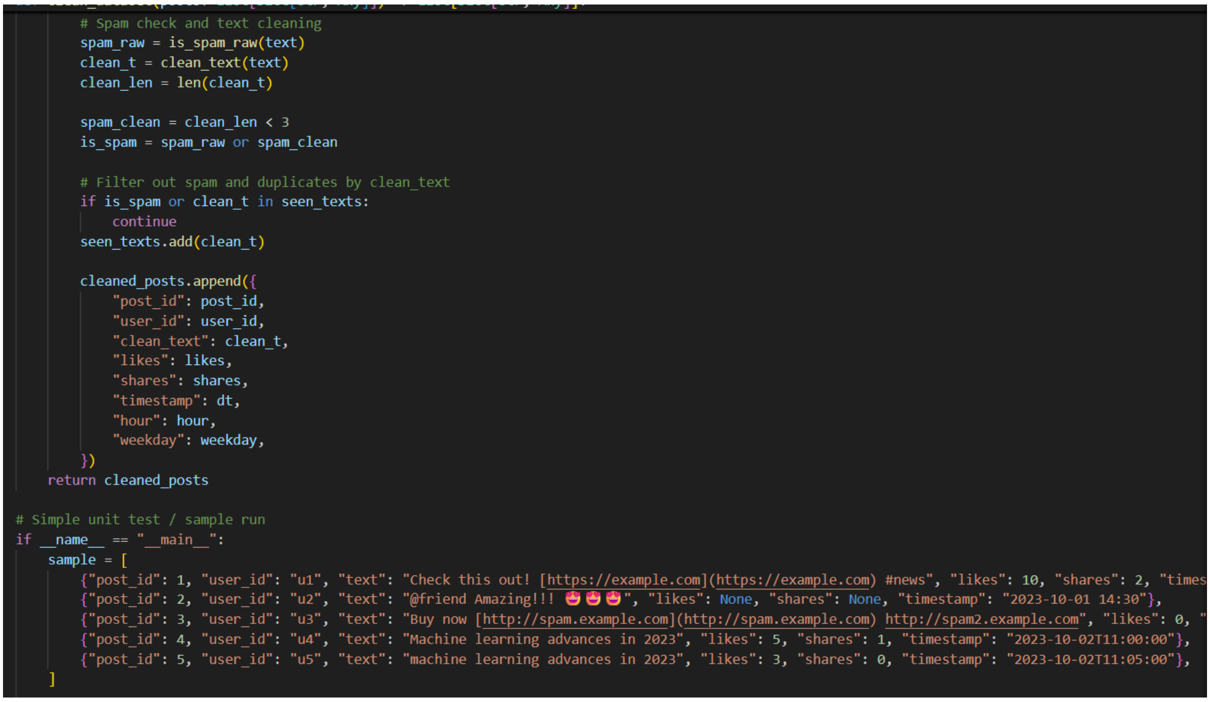
Code:



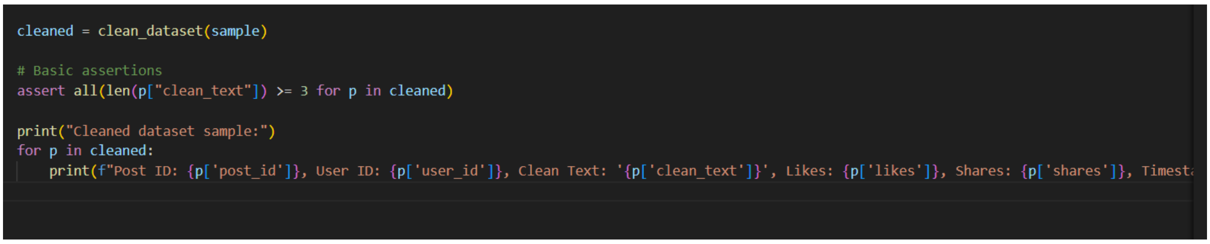
```python
import re
from datetime import datetime
from typing import List, Dict, Any

STOPWORDS = {
    "a","an","the","and","or","but","if","in","on","for","to","of","is","are","was","were",
    "it","this","that","with","as","by","at","from","be","has","have","had","i","you","we","they"
}
URL_RE = re.compile(r'https?://\S+|www\.\S+')
MENTION_RE = re.compile(r'@\w+')
EMOJI_RE = re.compile("[\U00010000-\U0010ffff]", flags=re.UNICODE)
PUNCT_RE = re.compile(r'[^a-z0-9\s]')

def clean_text(text: Any) -> str:
    if not isinstance(text, str):
        return ""
    s = text.lower()
    s = URL_RE.sub(" ", s)
    s = MENTION_RE.sub(" ", s)
    s = EMOJI_RE.sub(" ", s)
    s = s.replace("#", " ")
    s = PUNCT_RE.sub(" ", s)
    tokens = [t for t in s.split() if t and t not in STOPWORDS]
    return " ".join(tokens)

def is_spam_raw(text: Any) -> bool:
    if not isinstance(text, str):
        return True
    if len(URL_RE.findall(text)) > 1:
        return True
    if re.search(r'(.)\1{4,}', text):
        return True
    return False
```



```python
def parse_timestamp(ts_str: Any):
    if not isinstance(ts_str, str):
        return None, None, None
    for fmt in ("%Y-%m-%dT%H:%M:%S", "%Y-%m-%d %H:%M"):
        try:
            dt = datetime.strptime(ts_str, fmt)
            return dt, dt.hour, dt.strftime("%A")
        except ValueError:
            continue
    return None, None, None

def clean_dataset(posts: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
    cleaned_posts = []
    seen_texts = set()

    for post in posts:
        post_id = post.get("post_id")
        user_id = post.get("user_id")
        text = post.get("text", "")
        likes = post.get("likes")
        shares = post.get("shares")
        timestamp_str = post.get("timestamp")

        # Normalize likes/shares to int with fallback 0
        try:
            likes = int(likes) if likes is not None else 0
        except Exception:
            likes = 0
        try:
            shares = int(shares) if shares is not None else 0
        except Exception:
            shares = 0

        # Parse timestamp, hour, weekday
        dt, hour, weekday = parse_timestamp(timestamp_str)
```

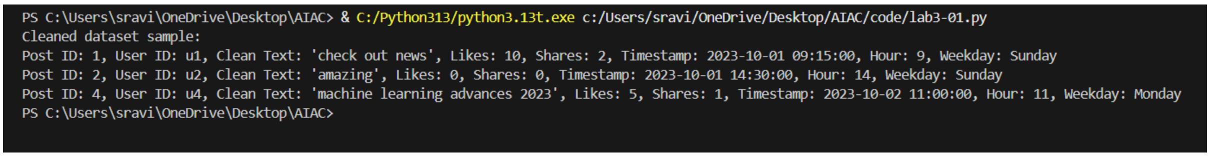


```python
        # Spam check and text cleaning
        spam_raw = is_spam_raw(text)
        clean_t = clean_text(text)
        clean_len = len(clean_t)

        spam_clean = clean_len < 3
        is_spam = spam_raw or spam_clean

        # Filter out spam and duplicates by clean_text
        if is_spam or clean_t in seen_texts:
            continue
        seen_texts.add(clean_t)

        cleaned_posts.append({
            "post_id": post_id,
            "user_id": user_id,
            "clean_text": clean_t,
            "likes": likes,
            "shares": shares,
            "timestamp": dt,
            "hour": hour,
            "weekday": weekday,
        })
    return cleaned_posts

# Simple unit test / sample run
if __name__ == "__main__":
    sample = [
        {"post_id": 1, "user_id": "u1", "text": "Check this out! [https://example.com](https://example.com) #news", "likes": 10, "shares": 2, "times
        {"post_id": 2, "user_id": "u2", "text": "@friend Amazing!!! 😍😍😍", "likes": None, "shares": None, "timestamp": "2023-10-01 14:30"},
        {"post_id": 3, "user_id": "u3", "text": "Buy now [http://spam.example.com](http://spam.example.com) http://spam2.example.com", "likes": 0, "
        {"post_id": 4, "user_id": "u4", "text": "Machine learning advances in 2023", "likes": 5, "shares": 1, "timestamp": "2023-10-02T11:00:00"},
        {"post_id": 5, "user_id": "u5", "text": "machine learning advances in 2023", "likes": 3, "shares": 0, "timestamp": "2023-10-02T11:05:00"},
    ]
```



```python
cleaned = clean_dataset(sample)

# Basic assertions
assert all(len(p["clean_text"]) >= 3 for p in cleaned)

print("Cleaned dataset sample:")
for p in cleaned:
    print(f"Post ID: {p['post_id']}, User ID: {p['user_id']}, Clean Text: '{p['clean_text']}', Likes: {p['likes']}, Shares: {p['shares']}, Timest
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/Python313/python3.13t.exe c:/Users/sravi/OneDrive/Desktop/AIAC/code/lab3-01.py
Cleaned dataset sample:
Post ID: 1, User ID: u1, Clean Text: 'check out news', Likes: 10, Shares: 2, Timestamp: 2023-10-01 09:15:00, Hour: 9, Weekday: Sunday
Post ID: 2, User ID: u2, Clean Text: 'amazing', Likes: 0, Shares: 0, Timestamp: 2023-10-01 14:30:00, Hour: 14, Weekday: Sunday
Post ID: 4, User ID: u4, Clean Text: 'machine learning advances 2023', Likes: 5, Shares: 1, Timestamp: 2023-10-02 11:00:00, Hour: 11, Weekday: Monday
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation of AI assistance used

- Prompted an AI assistant to suggest refactorings (the prompt above).
- AI suggested: add type hints, docstrings, input validation, and better modularization.
- Applied AI recommendations: improved regex clarity, explicit spam heuristics, defensive coding (type checks), and tiny unit-test assertions.

- Result: cleaner, maintainable script ready for integration into the Education-sector pipeline.

Q2:
Scenario: In the Agriculture sector, a company faces a challenge related to algorithms with
ai assistance.
Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

Prompt:

Implement a lightweight K-means clustering module in Python (no external libraries).

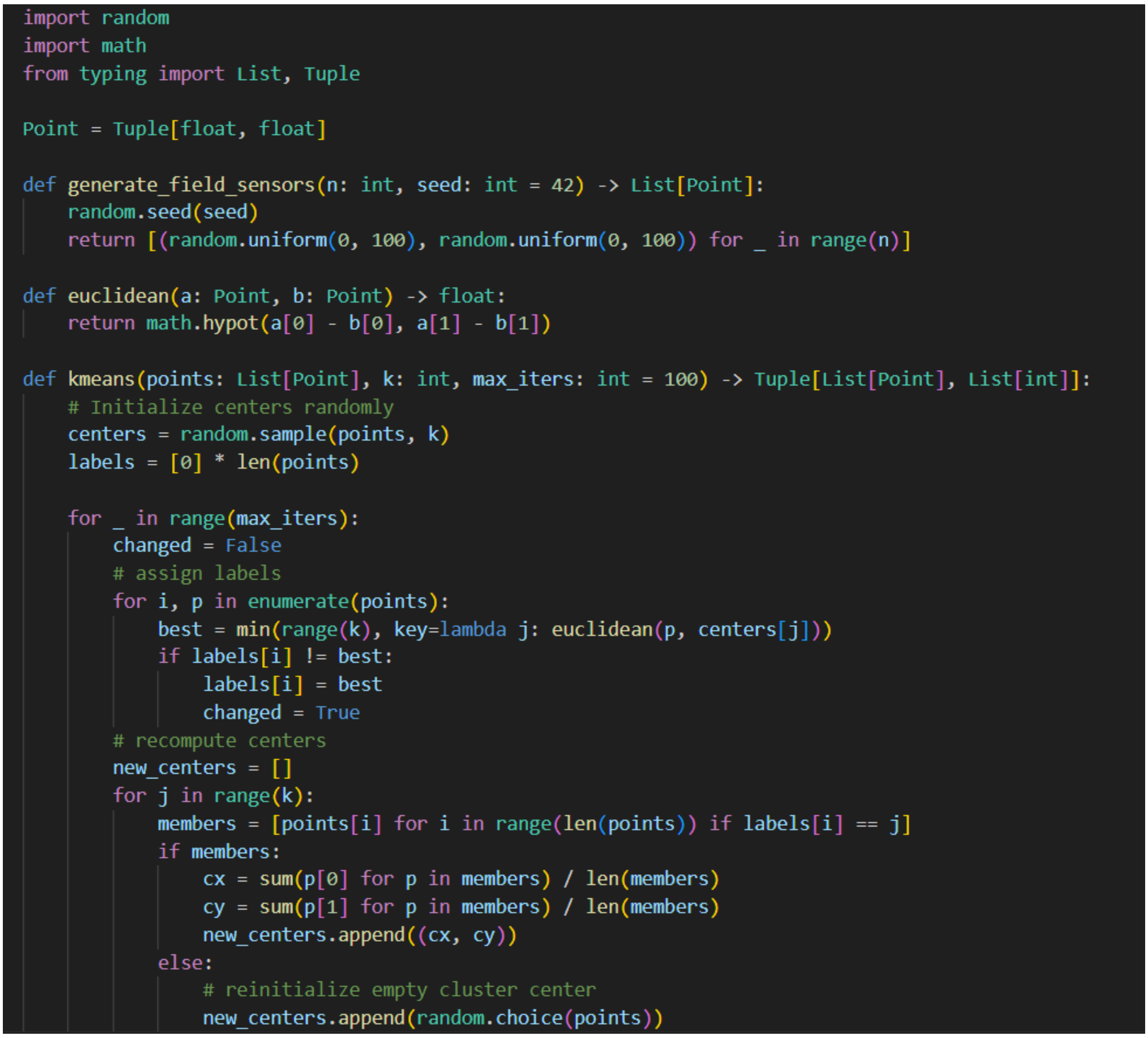
Use it to group field sensor coordinates into irrigation zones.

Provide functions: generate_field_sensors(n, seed), kmeans(points, k), wcss(...).

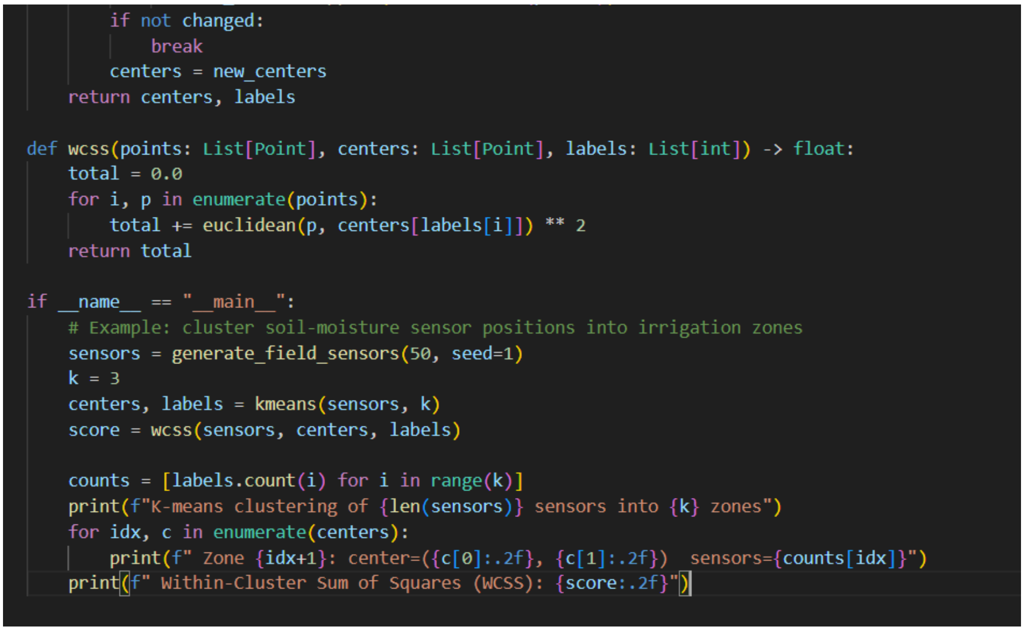
Include a small runnable example that prints cluster centers, member counts, and WCSS.

Keep code short and well-typed.

Code:

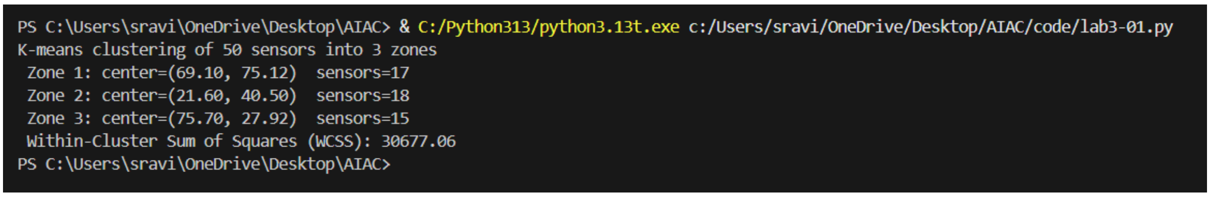


```python
import random
import math
from typing import List, Tuple

Point = Tuple[float, float]

def generate_field_sensors(n: int, seed: int = 42) -> List[Point]:
    random.seed(seed)
    return [(random.uniform(0, 100), random.uniform(0, 100)) for _ in range(n)]

def euclidean(a: Point, b: Point) -> float:
    return math.hypot(a[0] - b[0], a[1] - b[1])

def kmeans(points: List[Point], k: int, max_iters: int = 100) -> Tuple[List[Point], List[int]]:
    # Initialize centers randomly
    centers = random.sample(points, k)
    labels = [0] * len(points)

    for _ in range(max_iters):
        changed = False
        # assign labels
        for i, p in enumerate(points):
            best = min(range(k), key=lambda j: euclidean(p, centers[j]))
            if labels[i] != best:
                labels[i] = best
                changed = True
        # recompute centers
        new_centers = []
        for j in range(k):
            members = [points[i] for i in range(len(points)) if labels[i] == j]
            if members:
                cx = sum(p[0] for p in members) / len(members)
                cy = sum(p[1] for p in members) / len(members)
                new_centers.append((cx, cy))
            else:
                # reinitialize empty cluster center
                new_centers.append(random.choice(points))
```



```python
        if not changed:
            break
        centers = new_centers
    return centers, labels

def wcss(points: List[Point], centers: List[Point], labels: List[int]) -> float:
    total = 0.0
    for i, p in enumerate(points):
        total += euclidean(p, centers[labels[i]]) ** 2
    return total

if __name__ == "__main__":
    # Example: cluster soil-moisture sensor positions into irrigation zones
    sensors = generate_field_sensors(50, seed=1)
    k = 3
    centers, labels = kmeans(sensors, k)
    score = wcss(sensors, centers, labels)

    counts = [labels.count(i) for i in range(k)]
    print(f"K-means clustering of {len(sensors)} sensors into {k} zones")
    for idx, c in enumerate(centers):
        print(f" Zone {idx+1}: center=({c[0]:.2f}, {c[1]:.2f})  sensors={counts[idx]}")
    print(f" Within-Cluster Sum of Squares (WCSS): {score:.2f}")
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/Python313/python3.13t.exe c:/Users/sravi/OneDrive/Desktop/AIAC/code/lab3-01.py
K-means clustering of 50 sensors into 3 zones
 Zone 1: center=(69.10, 75.12)  sensors=17
 Zone 2: center=(21.60, 40.50)  sensors=18
 Zone 3: center=(75.70, 27.92)  sensors=15
 Within-Cluster Sum of Squares (WCSS): 30677.06
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation of AI assistance used

- Used an AI coding assistant to suggest a concise, standard K-means implementation and example harness.
- AI helped with function decomposition (euclidean, kmeans, wcss), initialization choices, and a simple stopping condition.
- I reviewed and adjusted the code for readability, determinism (seeded generator), and edge cases (empty cluster reinit).