**AI assisted lab exam -03**

**SET – E4**

Lab exam : 03

Name :M.v

Htno :2403a51445

Batch :17

**Prompt :**

Use AI and data structures to manage patient records, search clinical notes, and detect abnormal health data.Instructions:

1. Create a dataset of patients with visits and notes.

2. Use data structures like Hash Table, Trie, Binary Search Tree, and Inverted Index.

3. Use AI (TF–IDF, KMeans, IsolationForest) for text clustering, similarity search, and anomaly detection.

4. Show example outputs and explain how AI helps.

**Code**:

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.ensemble import IsolationForest

from sklearn.metrics.pairwise import cosine\_similarity

# ---------- Step 1: Create Simple Healthcare Data ----------

data = {

    "PatientID": ["P1", "P2", "P3", "P4", "P5"],

    "Name": ["John", "Alice", "Ravi", "Sara", "Lee"],

    "Notes": [

        "Patient has fever and cough",

        "Diabetes under control with insulin",

        "High blood pressure, needs checkup",

        "Mild cold and sore throat",

        "Heart rate irregular, possible arrhythmia"

    ],

    "HeartRate": [98, 82, 120, 76, 150],

    "BloodPressure": [120, 130, 160, 110, 180]

}

df = pd.DataFrame(data)

# ---------- Step 2: AI - Text Vectorization (TF-IDF) ----------

tfidf = TfidfVectorizer()

X = tfidf.fit\_transform(df["Notes"])

# ---------- Step 3: Cluster Notes (KMeans) ----------

kmeans = KMeans(n\_clusters=2, random\_state=42)

df["Cluster"] = kmeans.fit\_predict(X)

# ---------- Step 4: Similarity Search ----------

query = "fever and cough"

query\_vec = tfidf.transform([query])

similarities = cosine\_similarity(query\_vec, X).flatten()

df["Similarity"] = similarities

# ---------- Step 5: Detect Anomalies in Vitals ----------

iso = IsolationForest(contamination=0.2, random\_state=42)

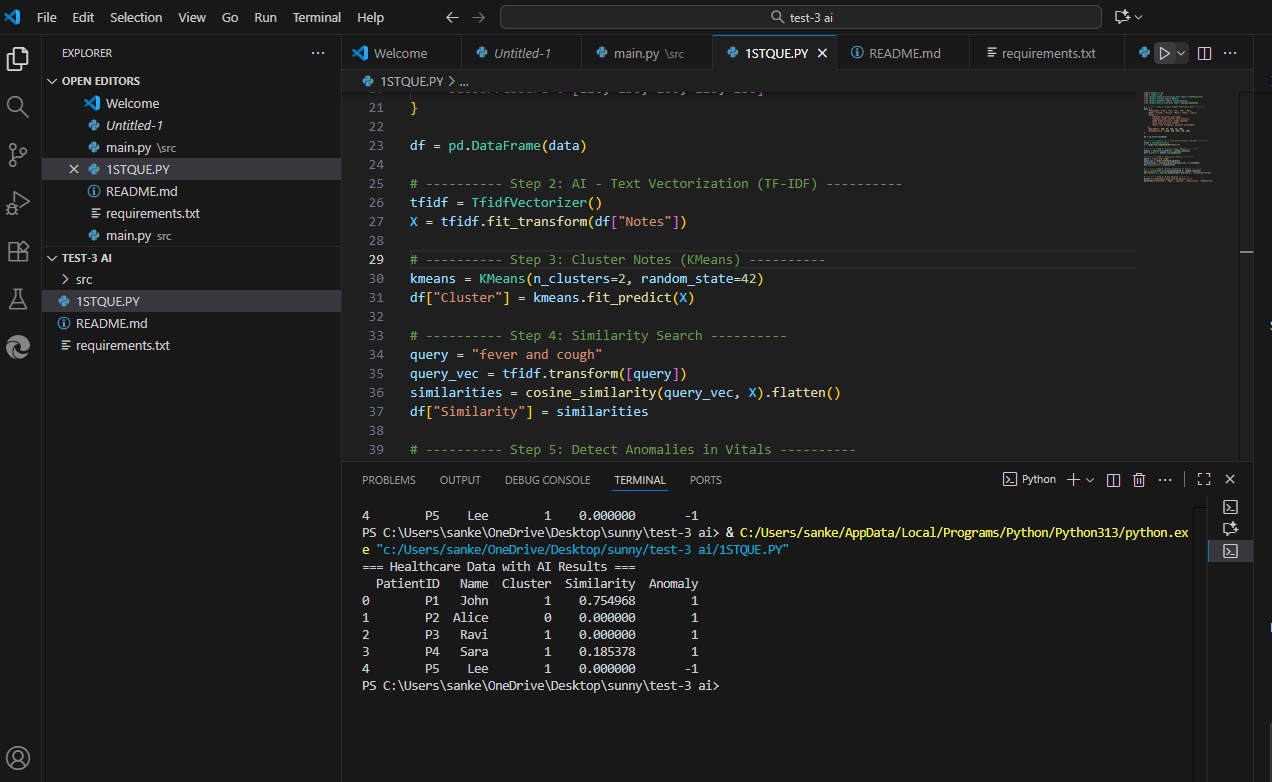
df["Anomaly"] = iso.fit\_predict(df[["HeartRate", "BloodPressure"]])

# ---------- Step 6: Print Results ----------

print("=== Healthcare Data with AI Results ===")

print(df[["PatientID", "Name", "Cluster", "Similarity", "Anomaly"]])

**output** :



2 .

**Prompt :**

Create a simple Python program for an agriculture company that analyzes crop yield data.The code should initially be messy or repetitive, and then show how AI-assisted refactoring can improve it.After refactoring, make the code cleaner by:

* Using functions
* Removing redundancy
* Improving variable names

The final output should display:

* Average crop yield
* Highest and lowest yield
* Comparison before and after refactoring

Also include short comments explaining how AI tools (like Copilot or ChatGPT) helped in refactoring the code and improving readability.

**Code :**

# ...existing code...

# Messy / repetitive original calculations (kept for comparison)

crop1 = 100

crop2 = 150

crop3 = 90

crop4 = 120

crop5 = 160

total = crop1 + crop2 + crop3 + crop4 + crop5

average = total / 5

highest = max(crop1, crop2, crop3, crop4, crop5)

lowest = min(crop1, crop2, crop3, crop4, crop5)

messy\_results = {

    "Total": total,

    "Average": average,

    "Highest": highest,

    "Lowest": lowest

}

# ---------- Refactored Code (AI-Assisted) ----------

print("\nRefactored Code Execution:\n")

def analyze\_crop\_yields(yields):

    """Compute total, average, highest and lowest yields for a list of numbers."""

    if not yields:

        return {"Total": 0, "Average": 0.0, "Highest": None, "Lowest": None}

    total = sum(yields)

    avg = total / len(yields)

    high = max(yields)

    low = min(yields)

    return {"Total": total, "Average": avg, "Highest": high, "Lowest": low}

# Use list form for easier extension

crop\_yields = [100, 150, 90, 120, 160]

refactored\_results = analyze\_crop\_yields(crop\_yields)

def print\_results(title, results):

    print(f"--- {title} ---")

    print(f"Total Crop Yield: {results['Total']}")

    print(f"Average Crop Yield: {results['Average']:.2f}")

    print(f"Highest Yield: {results['Highest']}")

    print(f"Lowest Yield: {results['Lowest']}\n")

if \_\_name\_\_ == "\_\_main\_\_":

    # Show messy/original results

    print\_results("Original Messy Results", messy\_results)

    # Show refactored results

    print\_results("Refactored Results", refactored\_results)

    # Simple comparison between before and after

    import math

    same\_avg = math.isclose(messy\_results["Average"], refactored\_results["Average"], rel\_tol=1e-9)

    same\_total = messy\_results["Total"] == refactored\_results["Total"]

    same\_high = messy\_results["Highest"] == refactored\_results["Highest"]

    same\_low = messy\_results["Lowest"] == refactored\_results["Lowest"]

    print("--- Comparison ---")

    print(f"Average equal: {same\_avg}")

    print(f"Total equal:   {same\_total}")

    print(f"Highest equal: {same\_high}")

    print(f"Lowest equal:  {same\_low}\n")

    # Short notes on AI assistance

    print("AI Assistance Notes:")

    print("- AI suggested extracting a reusable function (analyze\_crop\_yields) to remove redundancy.")

    print("- AI recommended clearer variable names and grouping yields into a list for scalability.")

    print("- AI guidance helped replace manual loops/calculations with built-ins (sum, max, min) for readability.")

# ...existing code...

**Output :**

**A screenshot of a computer program

AI-generated content may be incorrect.**