import streamlit as st

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

import io

# --- Page Config ---

st.set\_page\_config(page\_title="AHP Multi-Sector Evaluator", layout="wide")

# --- Constants ---

CRITERIA = [

    "Carbon Reduction Potential and Environmental Co-Benefits",

    "Economic Feasibility",

    "Technological Readiness and Implementation Feasibility",

    "Scalability and Long-Term Sustainability",

    "Policy Alignment",

    "Social Acceptance"

]

ALTERNATIVES = [

    "CCS / CCUS carbon capture and storage (filtre)",

    "Fuel Switching / Alternative Fuel Sources",

    "Reuse Waste Heat",

    "Sustainable Material Selection and Recycling",

    "Digitalization and Industry 4.0 Applications"

]

SECTORS = [

    "metal industries",

    "cement",

    "chemical production",

    "oil and gas",

    "critical mineral industry"

]

# --- AHP Logic ---

def pairwise\_matrix(items, session\_key):

    n = len(items)

    matrix = np.ones((n, n))

    for i in range(n):

        for j in range(i + 1, n):

            key = f"{session\_key}\_{i}\_{j}"

            val = st.slider(

                f"{items[i]} vs {items[j]}",

                1/9.0, 9.0, 1.0, step=0.1, key=key,

                help="Use values >1 to indicate more importance for the first item, <1 for the second."

            )

            matrix[i][j] = val

            matrix[j][i] = 1 / val

    return matrix

def normalize\_matrix(matrix):

    col\_sum = np.sum(matrix, axis=0)

    return matrix / col\_sum

def calculate\_priority\_vector(matrix):

    norm = normalize\_matrix(matrix)

    return np.mean(norm, axis=1)

def consistency\_ratio(matrix, priority\_vector):

    n = len(priority\_vector)

    lamda\_max = np.sum(np.dot(matrix, priority\_vector) / priority\_vector) / n

    ci = (lamda\_max - n) / (n - 1)

    RI\_dict = {1: 0.0, 2: 0.0, 3: 0.58, 4: 0.9, 5: 1.12, 6: 1.24, 7: 1.32, 8: 1.41, 9: 1.45, 10: 1.49}

    ri = RI\_dict.get(n, 1.49)

    cr = ci / ri if ri != 0 else 0

    return cr

# --- Streamlit UI ---

st.title("🌍 AHP Multi-Sector Decarbonization Evaluator")

st.markdown("""

Welcome to the AHP application for evaluating decarbonization strategies across sectors.

You'll walk through four steps to build a decision framework and receive a downloadable report.

""")

st.header("① Compare Evaluation Criteria")

with st.expander("Compare Criteria - Click to Expand"):

    st.write("Provide pairwise comparisons between the six evaluation criteria:")

    criteria\_matrix = pairwise\_matrix(CRITERIA, "criteria")

    criteria\_weights = calculate\_priority\_vector(criteria\_matrix)

    criteria\_cr = consistency\_ratio(criteria\_matrix, criteria\_weights)

    st.subheader("Criteria Weights")

    st.dataframe(pd.DataFrame({"Criteria": CRITERIA, "Weight": criteria\_weights}))

    st.markdown(f"\*\*Consistency Ratio (CR):\*\* `{criteria\_cr:.3f}`")

    if criteria\_cr > 0.1:

        st.warning("⚠️ The consistency ratio is high. Consider revisiting your judgments.")

st.header("② Sector-Wise Evaluation of Alternatives")

st.markdown("Evaluate alternatives under each sector by selecting below.")

selected\_sector = st.selectbox("Choose a Sector to Score Alternatives:", SECTORS)

if 'sector\_results' not in st.session\_state:

    st.session\_state.sector\_results = {}

sector\_best\_alternatives = {}

for criterion in CRITERIA:

    with st.expander(f"{selected\_sector.title()} - {criterion}"):

        matrix = pairwise\_matrix(ALTERNATIVES, f"{selected\_sector}\_{criterion}")

        weights = calculate\_priority\_vector(matrix)

        cr = consistency\_ratio(matrix, weights)

        df = pd.DataFrame({"Alternative": ALTERNATIVES, "Weight": weights})

        st.dataframe(df)

        st.markdown(f"\*\*Consistency Ratio (CR):\*\* `{cr:.3f}`")

        if cr > 0.1:

            st.warning("⚠️ Inconsistent comparison. Try adjusting the values.")

        st.session\_state.sector\_results[(selected\_sector, criterion)] = weights

st.header("③ Best Alternative per Sector")

sector\_final\_scores = {}

all\_sector\_scores = []

for sector in SECTORS:

    alt\_scores = np.zeros(len(ALTERNATIVES))

    for i, criterion in enumerate(CRITERIA):

        weights = st.session\_state.sector\_results.get((sector, criterion))

        if weights is not None:

            alt\_scores += criteria\_weights[i] \* weights

    sector\_final\_scores[sector] = alt\_scores

    best\_index = np.argmax(alt\_scores)

    sector\_best\_alternatives[sector] = ALTERNATIVES[best\_index]

    sector\_score\_df = pd.DataFrame({"Alternative": ALTERNATIVES, "Score": alt\_scores})

    sector\_score\_df["Sector"] = sector

    all\_sector\_scores.append(sector\_score\_df)

    st.success(f"✅ \*\*{sector.title()}\*\*: Best Alternative → \*\*{ALTERNATIVES[best\_index]}\*\*")

    fig, ax = plt.subplots(figsize=(8, 3))

    ax.barh(ALTERNATIVES, alt\_scores, color="skyblue")

    ax.set\_title(f"{sector.title()} - Alternative Scores")

    ax.set\_xlabel("Score")

    ax.xaxis.set\_major\_locator(ticker.MaxNLocator(nbins=5))

    st.pyplot(fig)

st.header("④ Final AHP Between Sectoral Winners")

final\_alts = [sector\_best\_alternatives[sec] for sec in SECTORS]

final\_matrix = pairwise\_matrix(final\_alts, "final")

final\_weights = calculate\_priority\_vector(final\_matrix)

final\_cr = consistency\_ratio(final\_matrix, final\_weights)

final\_df = pd.DataFrame({"Sector": SECTORS, "Best Alternative": final\_alts, "Weight": final\_weights})

st.dataframe(final\_df)

st.markdown(f"\*\*Final Consistency Ratio (CR):\*\* `{final\_cr:.3f}`")

if final\_cr > 0.1:

    st.warning("⚠️ High inconsistency in final AHP step.")

fig, ax = plt.subplots(figsize=(8, 3))

ax.bar(SECTORS, final\_weights, color="mediumseagreen")

ax.set\_title("Final AHP Results by Sector")

ax.set\_ylabel("Weight")

ax.set\_xticks(np.arange(len(SECTORS)))

ax.set\_xticklabels(SECTORS, rotation=15, ha="right")

st.pyplot(fig)

st.header("⑤ Download Evaluation Report")

combined\_df = pd.concat(all\_sector\_scores, ignore\_index=True)

final\_df["Final Weight"] = final\_weights

buffer = io.BytesIO()

with pd.ExcelWriter(buffer, engine='xlsxwriter') as writer:

    pd.DataFrame({"Criteria": CRITERIA, "Weight": criteria\_weights}).to\_excel(writer, sheet\_name='Criteria Weights', index=False)

    combined\_df.to\_excel(writer, sheet\_name='Sector Scores', index=False)

    final\_df.to\_excel(writer, sheet\_name='Final Evaluation', index=False)

st.download\_button(

    label="📥 Download Excel Report",

    data=buffer,

    file\_name="AHP\_MultiSector\_Report.xlsx",

    mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

)