import streamlit as st

import numpy as np

import pandas as pd

import io

# --- Global 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"

]

# --- Shared Functions (AHP Methods) ---

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}"

col1, col2, col3 = st.columns([4, 6, 4])

with st.container():

st.markdown(

"<div style='border: 2px solid #333333; border-radius: 6px; padding: 0px; margin-bottom: 5px;'>",

unsafe\_allow\_html=True,

)

with col1:

st.markdown(

f"<div style='text-align:center; border:1px solid #ccc; padding:4px; border-radius:6px; font-size:14px;'>"

f"{items[i]}</div>",

unsafe\_allow\_html=True

)

with col2:

selection = st.slider(

label=" ",

min\_value=-9,

max\_value=9,

value=0,

step=1,

key=key,

help="0 = eşit önem, pozitif = sağ daha önemli, negatif = sol daha önemli"

)

if selection == 0:

matrix[i][j] = 1

matrix[j][i] = 1

elif selection > 0:

matrix[i][j] = 1 / selection

matrix[j][i] = selection

else: # selection < 0

matrix[i][j] = abs(selection)

matrix[j][i] = 1 / abs(selection)

with col3:

st.markdown(

f"<div style='text-align:center; border:1px solid #ccc; padding:4px; border-radius:6px; font-size:14px;'>"

f"{items[j]}</div>",

unsafe\_allow\_html=True

)

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

# --- TOPSIS Functions ---

def normalize\_decision\_matrix(matrix):

# Her sütun için Euclidean norm ile normalizasyon

norm\_matrix = np.zeros\_like(matrix, dtype=float)

for j in range(matrix.shape[1]):

col = matrix[:, j]

denom = np.sqrt(np.sum(col\*\*2))

norm\_matrix[:, j] = col if denom == 0 else col / denom

return norm\_matrix

def weighted\_normalized\_matrix(norm\_matrix, weights):

return norm\_matrix \* weights

def ideal\_solutions(weighted\_matrix):

positive\_ideal = np.max(weighted\_matrix, axis=0)

negative\_ideal = np.min(weighted\_matrix, axis=0)

return positive\_ideal, negative\_ideal

def calculate\_topsis\_scores(decision\_matrix, criteria\_weights):

norm\_matrix = normalize\_decision\_matrix(decision\_matrix)

weighted\_matrix = weighted\_normalized\_matrix(norm\_matrix, criteria\_weights)

positive\_ideal, negative\_ideal = ideal\_solutions(weighted\_matrix)

dist\_positive = np.sqrt(np.sum((weighted\_matrix - positive\_ideal)\*\*2, axis=1))

dist\_negative = np.sqrt(np.sum((weighted\_matrix - negative\_ideal)\*\*2, axis=1))

scores = dist\_negative / (dist\_positive + dist\_negative)

return scores

# --- Page: AHP (Orijinal Kodunuz) ---

def load\_ahp\_page():

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

st.markdown("""

<div style='text-align: center;'>

Çok sektörlü decarbonizasyon stratejilerini değerlendirmek için AHP uygulamasına hoş geldiniz.

Aşağıdaki adımları izleyerek karar çerçevenizi oluşturabilirsiniz.

</div>

""", unsafe\_allow\_html=True)

# ① Kriterlerin Karşılaştırılması

st.header("① Kriterlerin Karşılaştırılması")

with st.expander("Kriterleri Karşılaştır - Açmak için tıklayın"):

st.write("Altı değerlendirme kriteri için çift yönlü karşılaştırma yapınız:")

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

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

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

st.subheader("Kriter Ağırlıkları")

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

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

if criteria\_cr > 0.1:

st.warning("⚠️ Tutarsız karşılaştırma. Lütfen değerleri gözden geçirin.")

# ② Sektör Bazında Alternatiflerin Değerlendirilmesi

st.header("② Sektör Bazında Alternatiflerin Değerlendirilmesi")

st.markdown("<div style='text-align:center;'>Tüm sektörler için alternatifleri değerlendirin:</div>", unsafe\_allow\_html=True)

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

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

sector\_best\_alternatives = {}

all\_sector\_scores = []

for sector in SECTORS:

st.subheader(f"{sector.title()}")

for criterion in CRITERIA:

with st.expander(f"{criterion}", expanded=False):

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

weights = calculate\_priority\_vector(matrix)

cr = consistency\_ratio(matrix, weights)

df = pd.DataFrame({"Alternatif": ALTERNATIVES, "Ağırlık": weights})

st.dataframe(df)

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

if cr > 0.1:

st.warning("⚠️ Tutarsız karşılaştırma. Lütfen değerleri gözden geçirin.")

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

# ③ Her Sektör için En İyi Alternatifin Belirlenmesi

st.header("③ Sektör Bazında En İyi Alternatifin Belirlenmesi")

sector\_final\_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({"Alternatif": ALTERNATIVES, "Skor": alt\_scores})

sector\_score\_df["Sektör"] = sector

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

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

# ④ Sektör Kazananları Arasında Son AHP Değerlendirmesi

st.header("④ Sektör Kazananları Arasında Son AHP Değerlendirmesi")

final\_alts = [f"{SECTORS[i].title()}: {alt}" for i, alt in enumerate([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({

"Sektör": SECTORS,

"En İyi Alternatif": [sector\_best\_alternatives[sec] for sec in SECTORS],

"Ağırlık": final\_weights

})

st.dataframe(final\_df)

st.markdown(f"\*\*Son Tutarlılık Oranı (CR):\*\* `{final\_cr:.3f}`")

if final\_cr > 0.1:

st.warning("⚠️ Final aşamasında yüksek tutarsızlık mevcut.")

# ⑤ Sonuçların Özetlenmesi ve Rapor İndirimi

st.header("🏆 En İyi Alternatif")

best\_final\_index = np.argmax(final\_weights)

worst\_final\_index = np.argmin(final\_weights)

st.success(f"🏅 En İyi Alternatif: \*\*{final\_alts[best\_final\_index]}\*\* (Skor: `{final\_weights[best\_final\_index]:.4f}`)")

st.info(f"🔻 En Düşük Skorlu Alternatif: \*\*{final\_alts[worst\_final\_index]}\*\* (Skor: `{final\_weights[worst\_final\_index]:.4f}`)")

st.header("⑤ Değerlendirme Raporunu İndir")

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

final\_df["Son Ağırlık"] = final\_weights

buffer = io.BytesIO()

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

pd.DataFrame({"Kriter": CRITERIA, "Ağırlık": criteria\_weights}).to\_excel(writer, sheet\_name='Kriter Ağırlıkları', index=False)

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

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

st.download\_button(

label="📥 Excel Raporunu İndir",

data=buffer,

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

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

)

# Ana sayfaya dönüş butonu

if st.button("Ana Sayfaya Dön"):

st.session\_state['page'] = 'main'

st.experimental\_rerun()

# --- Page: TOPSIS (Dönüştürülmüş Kod) ---

def load\_topsis\_page():

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

st.markdown("""

<div style='text-align: center;'>

TOPSIS uygulamasına hoş geldiniz.

Bu yöntemde, alternatifler sektör bazında 1-10 arası performans puanları ile değerlendirilmekte ve TOPSIS ile sıralanmaktadır.

</div>

""", unsafe\_allow\_html=True)

# ① Kriter Ağırlıkları (AHP ile)

st.header("① Kriterlerin Karşılaştırılması (AHP)")

with st.expander("Kriterleri Karşılaştır - Açmak için tıklayın"):

st.write("Altı değerlendirme kriteri için çift yönlü karşılaştırma yapınız:")

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

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

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

st.subheader("Kriter Ağırlıkları")

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

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

if criteria\_cr > 0.1:

st.warning("⚠️ Tutarsız karşılaştırma. Lütfen değerleri gözden geçirin.")

# ② Sektör Bazında Alternatiflerin Değerlendirilmesi (TOPSIS)

st.header("② Sektör Bazında Alternatiflerin Değerlendirilmesi (TOPSIS)")

st.markdown("<div style='text-align:center;'>Tüm sektörler için alternatifleri değerlendirin:</div>", unsafe\_allow\_html=True)

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

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

for sector in SECTORS:

st.subheader(f"{sector.title()}")

decision\_matrix = np.zeros((len(ALTERNATIVES), len(CRITERIA)))

for idx, criterion in enumerate(CRITERIA):

with st.expander(f"{criterion}", expanded=False):

st.write("Her alternatif için 1-10 arası performans puanı verin:")

scores = []

for alt in ALTERNATIVES:

score = st.slider(f"{alt}", min\_value=1, max\_value=10, value=5, key=f"{sector}\_{criterion}\_{alt}")

scores.append(score)

decision\_matrix[:, idx] = scores

st.session\_state.sector\_scores[sector] = decision\_matrix

# ③ Her Sektör için TOPSIS Sonuçları

st.header("③ Sektör Bazında En İyi Alternatifin Belirlenmesi (TOPSIS Sonuçları)")

sector\_best\_alternatives = {}

all\_sector\_topsis\_scores = []

for sector in SECTORS:

decision\_matrix = st.session\_state.sector\_scores.get(sector)

topsis\_scores = calculate\_topsis\_scores(decision\_matrix, criteria\_weights)

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

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

df = pd.DataFrame({"Alternatif": ALTERNATIVES, "TOPSIS Skoru": topsis\_scores})

df["Sektör"] = sector

all\_sector\_topsis\_scores.append(df)

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

st.dataframe(df)

# ④ Sektör Kazananları Arasında Final TOPSIS Değerlendirmesi

st.header("④ Sektör Kazananları Arasında Final TOPSIS Değerlendirmesi")

final\_alts = [f"{sector.title()}: {sector\_best\_alternatives[sector]}" for sector in SECTORS]

st.markdown("Final alternatifleri için iki son kriter üzerinden (Overall Effectiveness, Feasibility) puan verin:")

final\_criteria = ["Overall Effectiveness", "Feasibility"]

final\_decision\_matrix = np.zeros((len(final\_alts), len(final\_criteria)))

for i, alt in enumerate(final\_alts):

st.markdown(f"\*\*{alt}\*\*")

for j, crit in enumerate(final\_criteria):

score = st.slider(f"{crit}", min\_value=1, max\_value=10, value=5, key=f"final\_{i}\_{crit}")

final\_decision\_matrix[i, j] = score

final\_weights = np.array([0.5, 0.5])

final\_topsis\_scores = calculate\_topsis\_scores(final\_decision\_matrix, final\_weights)

final\_results\_df = pd.DataFrame({

"Sektör Kazananı": final\_alts,

"Final TOPSIS Skoru": final\_topsis\_scores

})

st.dataframe(final\_results\_df)

best\_final\_index = np.argmax(final\_topsis\_scores)

worst\_final\_index = np.argmin(final\_topsis\_scores)

st.success(f"🏅 En İyi Alternatif: \*\*{final\_alts[best\_final\_index]}\*\* (Skor: `{final\_topsis\_scores[best\_final\_index]:.4f}`)")

st.info(f"🔻 En Düşük Skorlu Alternatif: \*\*{final\_alts[worst\_final\_index]}\*\* (Skor: `{final\_topsis\_scores[worst\_final\_index]:.4f}`)")

# ⑤ Rapor İndirimi

st.header("⑤ Değerlendirme Raporunu İndir")

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

final\_results\_df["Final TOPSIS Skoru"] = final\_topsis\_scores

buffer = io.BytesIO()

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

pd.DataFrame({"Kriter": CRITERIA, "Ağırlık": criteria\_weights}).to\_excel(writer, sheet\_name='Kriter Ağırlıkları', index=False)

combined\_df.to\_excel(writer, sheet\_name='Sektör TOPSIS Skorları', index=False)

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

st.download\_button(

label="📥 Excel Raporunu İndir",

data=buffer,

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

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

)

# Ana sayfaya dönüş butonu

if st.button("Ana Sayfaya Dön"):

st.session\_state['page'] = 'main'

st.experimental\_rerun()

# --- Main Navigation Page ---

def main():

if 'page' not in st.session\_state:

st.session\_state['page'] = 'main'

if st.session\_state['page'] == 'main':

st.title("Decarbonization Evaluator")

st.markdown("\*\*Hangi yöntemi kullanmak istersiniz?\*\*")

col1, col2 = st.columns(2)

with col1:

if st.button("AHP"):

st.session\_state['page'] = 'ahp'

st.experimental\_rerun()

with col2:

if st.button("TOPSIS"):

st.session\_state['page'] = 'topsis'

st.experimental\_rerun()

elif st.session\_state['page'] == 'ahp':

load\_ahp\_page()

elif st.session\_state['page'] == 'topsis':

load\_topsis\_page()

if \_\_name\_\_ == '\_\_main\_\_':

main()