import streamlit as st

import pandas as pd

import numpy as np

import random

import plotly.express as px

import plotly.graph\_objects as go

from datetime import datetime

# Set page config

st.set\_page\_config(

page\_title="Poker AI Full Tutorial & Predictor",

page\_icon="🃏",

layout="wide"

)

# Constants

SUITS = {'S': '♠', 'H': '♥', 'D': '♦', 'C': '♣'}

RANKS = {'A': 'A', '2': '2', '3': '3', '4': '4', '5': '5', '6': '6', '7': '7', '8': '8', '9': '9', 'T': '10', 'J': 'J', 'Q': 'Q', 'K': 'K'}

# Tutorial constants

HAND\_NAMES = {

0: 'High Card', 1: 'One Pair', 2: 'Two Pairs', 3: 'Three of a Kind',

4: 'Straight', 5: 'Flush', 6: 'Full House', 7: 'Four of a Kind',

8: 'Straight Flush', 9: 'Royal Flush'

}

HAND\_STRENGTH = {

0: 0.05, 1: 0.15, 2: 0.25, 3: 0.35, 4: 0.50,

5: 0.60, 6: 0.75, 7: 0.90, 8: 0.95, 9: 0.99

}

class RealPokerDataProcessor:

def \_\_init\_\_(self):

self.dataset = None

self.model = None

self.is\_trained = False

self.training\_accuracy = 0.0

def load\_dataset(self, uploaded\_file):

try:

self.dataset = pd.read\_csv(uploaded\_file)

return True

except Exception as e:

st.error(f"Error loading dataset: {e}")

return False

def create\_demo\_dataset(self, n\_samples=1000):

demo\_data = []

for \_ in range(n\_samples):

hole\_cards = self.\_generate\_random\_cards(2)

board\_cards = self.\_generate\_random\_cards(5, exclude=hole\_cards)

hand\_strength = self.\_evaluate\_hand\_strength(hole\_cards + board\_cards)

row = {

'hole\_card\_1': hole\_cards[0], 'hole\_card\_2': hole\_cards[1],

'board\_card\_1': board\_cards[0], 'board\_card\_2': board\_cards[1], 'board\_card\_3': board\_cards[2],

'board\_card\_4': board\_cards[3], 'board\_card\_5': board\_cards[4], 'strength': hand\_strength

}

demo\_data.append(row)

self.dataset = pd.DataFrame(demo\_data)

return True

def \_generate\_random\_cards(self, n\_cards, exclude=None):

if exclude is None: exclude = []

all\_cards = [f"{r}{s}" for s in SUITS.keys() for r in RANKS.keys()]

available = [c for c in all\_cards if c not in exclude]

return random.sample(available, n\_cards)

def \_evaluate\_hand\_strength(self, cards):

ranks = [c[0] for c in cards]

suits = [c[1] for c in cards]

counts = sorted([ranks.count(r) for r in set(ranks)], reverse=True)

if counts[0] >= 4: return random.randint(85, 95)

if counts[0] >= 3 and counts[1] >= 2: return random.randint(80, 90)

if max([suits.count(s) for s in set(suits)]) >= 5: return random.randint(70, 85)

if counts[0] >= 3: return random.randint(60, 75)

if counts[0] >= 2 and counts[1] >= 2: return random.randint(45, 65)

if counts[0] >= 2: return random.randint(25, 50)

return random.randint(10, 35)

def prepare\_features(self):

rank\_map = {'A': 14, 'K': 13, 'Q': 12, 'J': 11, 'T': 10, '9': 9, '8': 8, '7': 7, '6': 6, '5': 5, '4': 4, '3': 3, '2': 2}

suit\_map = {'S': 1, 'H': 2, 'D': 3, 'C': 4}

features, targets = [], []

for \_, row in self.dataset.iterrows():

hole = [row['hole\_card\_1'], row['hole\_card\_2']]

board = [row[f'board\_card\_{i}'] for i in range(1, 6)]

cards = hole + board

feature = []

for c in cards:

feature.extend([rank\_map[c[0]], suit\_map[c[1]]])

features.append(feature)

targets.append(row['strength'] / 100)

return np.array(features), np.array(targets)

def train\_model(self):

X, y = self.prepare\_features()

if len(X) == 0: return False

X\_bias = np.column\_stack([np.ones(len(X)), X])

self.model = np.linalg.lstsq(X\_bias, y, rcond=None)[0]

preds = np.clip(X\_bias @ self.model, 0, 1)

self.training\_accuracy = 1 - np.sum((y - preds)\*\*2) / np.sum((y - y.mean())\*\*2)

self.is\_trained = True

return True

def predict(self, hole, board):

if not self.is\_trained: return 0.5

rank\_map = {'A': 14, 'K': 13, 'Q': 12, 'J': 11, 'T': 10, '9': 9, '8': 8, '7': 7, '6': 6, '5': 5, '4': 4, '3': 3, '2': 2}

suit\_map = {'S': 1, 'H': 2, 'D': 3, 'C': 4}

cards = hole + board

features = []

for c in cards:

features.extend([rank\_map.get(c[0], 2), suit\_map.get(c[1], 1)])

X = np.hstack(([1], features))

pred = np.dot(X, self.model)

return np.clip(pred, 0, 1)

def parse\_card(card\_str):

return f"{RANKS.get(card\_str[0], card\_str[0])}{SUITS.get(card\_str[1], card\_str[1])}" if len(card\_str) == 2 else card\_str

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

st.session\_state.processor = RealPokerDataProcessor()

if 'hands\_analyzed' not in st.session\_state:

st.session\_state.hands\_analyzed = 0

st.session\_state.correct\_predictions = 0

st.session\_state.progress\_history = []

st.title("🃏 Poker AI Full Tutorial & Predictor")

with st.sidebar:

uploaded\_file = st.file\_uploader("Upload Dataset (CSV)", type=['csv'])

if uploaded\_file and st.button("Load Dataset"):

if st.session\_state.processor.load\_dataset(uploaded\_file):

st.success("Dataset loaded!")

if st.button("Use Demo Dataset"):

st.session\_state.processor.create\_demo\_dataset()

st.success("Demo dataset created!")

if st.session\_state.processor.dataset is not None and not st.session\_state.processor.is\_trained:

if st.button("Train Model"):

if st.session\_state.processor.train\_model():

st.success(f"Model trained! Accuracy: {st.session\_state.processor.training\_accuracy:.1%}")

st.metric("Hands Analyzed", st.session\_state.hands\_analyzed)

skill\_level = st.selectbox("Teaching Level", ["beginner", "intermediate", "advanced"])

if st.session\_state.processor.dataset is not None:

tabs = st.tabs(["Analyzer", "Dataset", "Tutorial", "Practice", "Analytics"])

with tabs[0]:

st.header("Analyze Poker Hand")

col1, col2 = st.columns(2)

with col1:

st.write("Hole Cards:")

hole1\_suit = st.selectbox("Hole Card 1 Suit", options=list(SUITS.keys()), format\_func=lambda s: SUITS[s], key="hole1\_suit")

hole1\_rank = st.selectbox("Hole Card 1 Rank", options=list(RANKS.keys()), format\_func=lambda r: RANKS[r], key="hole1\_rank")

hole2\_suit = st.selectbox("Hole Card 2 Suit", options=list(SUITS.keys()), format\_func=lambda s: SUITS[s], key="hole2\_suit")

hole2\_rank = st.selectbox("Hole Card 2 Rank", options=list(RANKS.keys()), format\_func=lambda r: RANKS[r], key="hole2\_rank")

hole1 = f"{hole1\_rank}{hole1\_suit}"

hole2 = f"{hole2\_rank}{hole2\_suit}"

st.write("Board Cards:")

board = []

for i in range(5):

suit = st.selectbox(f"Board Card {i+1} Suit", options=list(SUITS.keys()), format\_func=lambda s: SUITS[s], key=f"board\_suit\_{i}")

rank = st.selectbox(f"Board Card {i+1} Rank", options=list(RANKS.keys()), format\_func=lambda r: RANKS[r], key=f"board\_rank\_{i}")

board.append(f"{rank}{suit}")

if st.button("Analyze Hand"):

win\_prob = st.session\_state.processor.predict([hole1, hole2], board)

st.session\_state.result = (hole1, hole2, board, win\_prob)

if 'result' in st.session\_state:

hole1, hole2, board, win\_prob = st.session\_state.result

st.write(f"Your Hand: {parse\_card(hole1)} {parse\_card(hole2)}")

st.write(f"Board: {' '.join([parse\_card(c) for c in board if c])}")

st.metric("Win Probability", f"{win\_prob:.1%}")

st.progress(win\_prob, text=f"{win\_prob:.1%}")

st.session\_state.hands\_analyzed += 1

with tabs[1]:

st.header("Dataset Overview")

st.write(st.session\_state.processor.dataset.head())

st.write("Total Hands:", len(st.session\_state.processor.dataset))

if 'strength' in st.session\_state.processor.dataset.columns:

fig = px.histogram(st.session\_state.processor.dataset, x='strength', nbins=20, title='Hand Strength Distribution')

st.plotly\_chart(fig)

with tabs[2]:

st.header("Poker Hand Rankings Tutorial")

for rank in range(9, -1, -1):

strength = HAND\_STRENGTH[rank]

st.write(f"\*\*{10-rank}. {HAND\_NAMES[rank]}\*\* - {strength:.0%} strength")

st.write("---")

st.subheader("Quick Quiz")

if st.button("Generate Quiz Hand"):

quiz\_hand = [(random.choice(list(SUITS.keys())), random.randint(1, 13)) for \_ in range(5)]

st.session\_state.quiz\_hand = quiz\_hand

if 'quiz\_hand' in st.session\_state:

quiz\_display = " ".join([f"{rank}{suit}" for suit, rank in st.session\_state.quiz\_hand])

st.write(f"Hand: {quiz\_display}")

with tabs[3]:

st.header("Practice Mode")

if st.button("Generate Random Practice Hand"):

hole\_cards = st.session\_state.processor.\_generate\_random\_cards(2)

board\_cards = st.session\_state.processor.\_generate\_random\_cards(5, exclude=hole\_cards)

st.session\_state.practice = (hole\_cards, board\_cards)

if 'practice' in st.session\_state:

hole\_cards, board\_cards = st.session\_state.practice

st.write("Your Hand:", ' '.join([parse\_card(c) for c in hole\_cards]))

st.write("Board:", ' '.join([parse\_card(c) for c in board\_cards]))

guess = st.slider("Estimate Win Probability (%)", 0, 100, 50)

if st.button("Check Answer"):

actual = st.session\_state.processor.predict(hole\_cards, board\_cards)

st.metric("Your Estimate", f"{guess}%")

st.metric("Actual", f"{actual\*100:.1f}%")

with tabs[4]:

st.header("Analytics")

st.metric("Total Hands Analyzed", st.session\_state.hands\_analyzed)