import streamlit as st

import pandas as pd

import numpy as np

import random

from datetime import datetime

import plotly.express as px

import plotly.graph\_objects as go

# Set page config

st.set\_page\_config(

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

page\_icon="🃏",

layout="wide"

)

# Constants

SUITS = {1: '♥', 2: '♠', 3: '♦', 4: '♣'}

RANKS = {1: 'A', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8',

9: '9', 10: '10', 11: 'J', 12: 'Q', 13: 'K'}

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

}

def evaluate\_poker\_hand(cards):

"""Evaluate poker hand and return class (0-9)"""

suits = [card[0] for card in cards]

ranks = sorted([card[1] for card in cards])

# Count rank frequencies

rank\_counts = {}

for rank in ranks:

rank\_counts[rank] = rank\_counts.get(rank, 0) + 1

counts = sorted(rank\_counts.values(), reverse=True)

# Check for flush

is\_flush = len(set(suits)) == 1

# Check for straight

is\_straight = False

if ranks == list(range(ranks[0], ranks[0] + 5)):

is\_straight = True

elif ranks == [1, 10, 11, 12, 13]: # A-10-J-Q-K

is\_straight = True

# Classify hand

if is\_straight and is\_flush:

if ranks == [1, 10, 11, 12, 13] or ranks == [9, 10, 11, 12, 13]:

return 9 # Royal Flush

else:

return 8 # Straight Flush

elif counts[0] == 4:

return 7 # Four of a Kind

elif counts[0] == 3 and counts[1] == 2:

return 6 # Full House

elif is\_flush:

return 5 # Flush

elif is\_straight:

return 4 # Straight

elif counts[0] == 3:

return 3 # Three of a Kind

elif counts[0] == 2 and counts[1] == 2:

return 2 # Two Pairs

elif counts[0] == 2:

return 1 # One Pair

else:

return 0 # High Card

def predict\_action(hand\_strength, pot\_odds):

"""Simple rule-based action prediction"""

if hand\_strength > 0.7:

return 2, [0.1, 0.2, 0.7] # Raise

elif hand\_strength > pot\_odds:

return 1, [0.2, 0.6, 0.2] # Call

else:

return 0, [0.7, 0.2, 0.1] # Fold

def calculate\_win\_probability(hand\_class, num\_opponents=1):

"""Calculate win probability based on hand strength"""

base\_strength = HAND\_STRENGTH[hand\_class]

win\_prob = base\_strength \*\* num\_opponents

if hand\_class >= 7: # Strong hands

win\_prob = max(0.85, win\_prob)

elif hand\_class <= 1: # Weak hands

win\_prob = min(0.25, win\_prob)

return win\_prob

def generate\_explanation(hand\_class, cards, skill\_level="beginner"):

"""Generate educational explanation"""

hand\_name = HAND\_NAMES[hand\_class]

card\_str = " ".join([f"{RANKS[rank]}{SUITS[suit]}" for suit, rank in cards])

if skill\_level == "beginner":

explanations = {

0: f"🃏 \*\*{hand\_name}\*\*: You have no pairs. Your highest card matters most.",

1: f"👥 \*\*{hand\_name}\*\*: Two cards of the same rank! Better than high card.",

2: f"👥👥 \*\*{hand\_name}\*\*: Two different pairs! A decent hand.",

3: f"🎯 \*\*{hand\_name}\*\*: Three cards of the same rank! Strong hand.",

4: f"📈 \*\*{hand\_name}\*\*: Five consecutive cards! Very strong.",

5: f"🌈 \*\*{hand\_name}\*\*: Five cards same suit! Strong hand.",

6: f"🏠 \*\*{hand\_name}\*\*: Three of a kind + pair! Excellent!",

7: f"🔥 \*\*{hand\_name}\*\*: Four same rank! Almost unbeatable!",

8: f"💎 \*\*{hand\_name}\*\*: Straight + flush! Nearly unbeatable!",

9: f"👑 \*\*{hand\_name}\*\*: Best possible hand! You cannot lose!"

}

else: # Intermediate/Advanced

strength = HAND\_STRENGTH[hand\_class]

explanations = {

0: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Play tight.",

1: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Value vs weak hands.",

2: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Good for value betting.",

3: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Strong - bet for value.",

4: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Very strong hand.",

5: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Premium hand.",

6: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Excellent hand.",

7: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Monster hand.",

8: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Nearly nuts.",

9: f"📊 \*\*{hand\_name}\*\* ({card\_str}): {strength:.0%} strength. Absolute nuts."

}

return explanations.get(hand\_class, "Unknown hand")

def generate\_action\_explanation(action, hand\_strength, pot\_odds):

"""Generate explanation for recommended action"""

actions = ["Fold", "Call", "Raise"]

recommended = actions[action]

if action == 0: # Fold

return f"🎯 \*\*Recommended: {recommended}\*\*\n\n💡 Your hand strength ({hand\_strength:.1%}) is below pot odds ({pot\_odds:.1%}). Save chips for better spots!"

elif action == 1: # Call

return f"🎯 \*\*Recommended: {recommended}\*\*\n\n💡 Your hand strength ({hand\_strength:.1%}) justifies the pot odds ({pot\_odds:.1%}). Reasonable call."

else: # Raise

return f"🎯 \*\*Recommended: {recommended}\*\*\n\n💡 Strong hand ({hand\_strength:.1%})! Bet for value to build the pot."

# Initialize session state

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.session\_state.current\_hand = None

# App header

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

st.markdown("\*Learn poker with AI-powered hand analysis and decision-making guidance\*")

# Sidebar

with st.sidebar:

st.header("📊 Your Progress")

accuracy = (st.session\_state.correct\_predictions / max(1, st.session\_state.hands\_analyzed)) \* 100

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

st.metric("Accuracy", f"{accuracy:.1f}%")

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

if st.button("🔄 Reset Progress"):

st.session\_state.hands\_analyzed = 0

st.session\_state.correct\_predictions = 0

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

st.rerun()

# Main tabs

tab1, tab2, tab3, tab4 = st.tabs(["🎮 Hand Analyzer", "📚 Tutorial", "🎲 Practice", "📊 Analytics"])

with tab1:

st.header("🎮 Poker Hand Analyzer")

col1, col2 = st.columns([1, 1])

with col1:

st.subheader("🃏 Input Your Hand")

input\_method = st.radio("Input Method", ["Manual Entry", "Random Hand", "Example Hands"])

if input\_method == "Manual Entry":

st.write("Enter 5 cards:")

cards = []

for i in range(5):

col\_suit, col\_rank = st.columns(2)

with col\_suit:

suit = st.selectbox(f"Suit {i+1}", [1, 2, 3, 4],

format\_func=lambda x: f"{SUITS[x]}", key=f"suit\_{i}")

with col\_rank:

rank = st.selectbox(f"Rank {i+1}", list(range(1, 14)),

format\_func=lambda x: RANKS[x], key=f"rank\_{i}")

cards.append((suit, rank))

st.session\_state.current\_hand = cards

elif input\_method == "Random Hand":

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

cards = []

used\_cards = set()

while len(cards) < 5:

suit = random.randint(1, 4)

rank = random.randint(1, 13)

card = (suit, rank)

if card not in used\_cards:

cards.append(card)

used\_cards.add(card)

st.session\_state.current\_hand = cards

else: # Example Hands

examples = {

"Royal Flush": [(1, 1), (1, 10), (1, 11), (1, 12), (1, 13)],

"Four of a Kind": [(1, 8), (2, 8), (3, 8), (4, 8), (1, 2)],

"Full House": [(1, 9), (2, 9), (3, 9), (1, 5), (2, 5)],

"Flush": [(2, 3), (2, 7), (2, 9), (2, 11), (2, 13)],

"Straight": [(1, 5), (2, 6), (3, 7), (4, 8), (1, 9)],

"Two Pairs": [(1, 8), (2, 8), (3, 3), (4, 3), (1, 12)],

"One Pair": [(1, 10), (2, 10), (3, 5), (4, 7), (1, 2)]

}

selected\_example = st.selectbox("Choose Example", list(examples.keys()))

if st.button("Load Example"):

st.session\_state.current\_hand = examples[selected\_example]

with col2:

st.subheader("🔍 Analysis Results")

if st.session\_state.current\_hand:

cards = st.session\_state.current\_hand

# Display cards

card\_display = " ".join([f"{RANKS[rank]}{SUITS[suit]}" for suit, rank in cards])

st.markdown(f"### {card\_display}")

# Evaluate hand

hand\_class = evaluate\_poker\_hand(cards)

hand\_name = HAND\_NAMES[hand\_class]

hand\_strength = HAND\_STRENGTH[hand\_class]

# Display results

st.success(f"\*\*Hand Type:\*\* {hand\_name}")

st.progress(hand\_strength, text=f"Hand Strength: {hand\_strength:.0%}")

# Win probability

win\_prob = calculate\_win\_probability(hand\_class)

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

# Tutorial explanation

explanation = generate\_explanation(hand\_class, cards, skill\_level)

st.info(explanation)

# Action recommendation

st.write("\*\*🎯 Action Recommendation:\*\*")

pot\_size = st.number\_input("Pot Size ($)", min\_value=1.0, value=50.0, step=5.0)

bet\_size = st.number\_input("Bet to Call ($)", min\_value=1.0, value=20.0, step=5.0)

if bet\_size < pot\_size:

pot\_odds = bet\_size / (pot\_size + bet\_size)

action, action\_probs = predict\_action(hand\_strength, pot\_odds)

# Display recommendation

actions = ["Fold 👎", "Call 📞", "Raise 📈"]

colors = ["red", "orange", "green"]

for i, (act, prob) in enumerate(zip(actions, action\_probs)):

if i == action:

st.success(f"\*\*{act}\*\*: {prob:.1%} confidence")

else:

st.write(f"{act}: {prob:.1%}")

# Explanation

action\_explanation = generate\_action\_explanation(action, hand\_strength, pot\_odds)

st.info(action\_explanation)

# Update progress

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

st.session\_state.progress\_history.append({

'hand': st.session\_state.hands\_analyzed,

'hand\_type': hand\_name,

'strength': hand\_strength,

'timestamp': datetime.now()

})

else:

st.info("👆 Select a hand input method above to get started!")

with tab2:

st.header("📚 Poker Tutorial")

lesson = st.selectbox(

"Choose Lesson",

["Hand Rankings", "Pot Odds", "Position Strategy", "Betting Basics"]

)

if lesson == "Hand Rankings":

st.subheader("🃏 Poker Hand Rankings (Strongest to Weakest)")

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

strength = HAND\_STRENGTH[rank]

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

# Interactive quiz

st.write("---")

st.subheader("🎯 Hand Recognition Quiz")

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

quiz\_cards = []

used\_cards = set()

while len(quiz\_cards) < 5:

suit = random.randint(1, 4)

rank = random.randint(1, 13)

card = (suit, rank)

if card not in used\_cards:

quiz\_cards.append(card)

used\_cards.add(card)

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

st.session\_state.quiz\_answer = evaluate\_poker\_hand(quiz\_cards)

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

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

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

user\_guess = st.selectbox("What hand is this?", list(HAND\_NAMES.values()))

if st.button("Check Answer"):

correct\_answer = HAND\_NAMES[st.session\_state.quiz\_answer]

if user\_guess == correct\_answer:

st.success("✅ Correct!")

st.session\_state.correct\_predictions += 1

else:

st.error(f"❌ Wrong. Correct answer: {correct\_answer}")

explanation = generate\_explanation(st.session\_state.quiz\_answer, st.session\_state.quiz\_hand, skill\_level)

st.info(explanation)

elif lesson == "Pot Odds":

st.subheader("🧮 Understanding Pot Odds")

st.write("""

\*\*Pot Odds\*\* help you decide if a call is profitable:

📊 \*\*Formula:\*\* Pot Odds = Bet to Call ÷ (Total Pot + Bet to Call)

🎯 \*\*Rule:\*\* Call when your winning chances > Pot Odds percentage

""")

# Calculator

st.write("\*\*Calculator:\*\*")

calc\_pot = st.number\_input("Current Pot", min\_value=1, value=100)

calc\_bet = st.number\_input("Bet to Call", min\_value=1, value=25)

calc\_odds = calc\_bet / (calc\_pot + calc\_bet)

st.metric("Pot Odds", f"{calc\_odds:.1%}")

st.write(f"You need at least {calc\_odds:.1%} chance of winning to call profitably")

else:

st.info(f"📚 {lesson} lesson coming soon!")

with tab3:

st.header("🎲 Practice Mode")

st.write("Test your poker skills!")

if st.button("🎯 New Practice Hand"):

practice\_cards = []

used\_cards = set()

while len(practice\_cards) < 5:

suit = random.randint(1, 4)

rank = random.randint(1, 13)

card = (suit, rank)

if card not in used\_cards:

practice\_cards.append(card)

used\_cards.add(card)

practice\_pot = random.randint(20, 200)

practice\_bet = random.randint(5, practice\_pot // 2)

st.session\_state.practice\_scenario = {

'cards': practice\_cards,

'pot': practice\_pot,

'bet': practice\_bet,

'hand\_class': evaluate\_poker\_hand(practice\_cards)

}

if 'practice\_scenario' in st.session\_state:

scenario = st.session\_state.practice\_scenario

# Display scenario

card\_display = " ".join([f"{RANKS[rank]}{SUITS[suit]}" for suit, rank in scenario['cards']])

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

st.write(f"\*\*Pot:\*\* ${scenario['pot']} | \*\*Bet to Call:\*\* ${scenario['bet']}")

# Get user predictions

col1, col2 = st.columns(2)

with col1:

st.write("\*\*What hand do you have?\*\*")

hand\_guess = st.selectbox("Hand Type", list(HAND\_NAMES.values()), key="hand\_guess")

with col2:

st.write("\*\*What should you do?\*\*")

action\_guess = st.selectbox("Action", ["Fold", "Call", "Raise"], key="action\_guess")

if st.button("Submit Answers"):

# Check hand recognition

correct\_hand = HAND\_NAMES[scenario['hand\_class']]

hand\_correct = hand\_guess == correct\_hand

# Check action

hand\_strength = HAND\_STRENGTH[scenario['hand\_class']]

pot\_odds = scenario['bet'] / (scenario['pot'] + scenario['bet'])

optimal\_action, \_ = predict\_action(hand\_strength, pot\_odds)

action\_names = ["Fold", "Call", "Raise"]

correct\_action = action\_names[optimal\_action]

action\_correct = action\_guess == correct\_action

# Show results

if hand\_correct:

st.success(f"✅ Hand: Correct! It's {correct\_hand}")

else:

st.error(f"❌ Hand: Wrong. It's {correct\_hand}")

if action\_correct:

st.success(f"✅ Action: Correct! {correct\_action} is optimal")

else:

st.error(f"❌ Action: Suboptimal. Better choice: {correct\_action}")

# Show explanations

explanation = generate\_explanation(scenario['hand\_class'], scenario['cards'], skill\_level)

st.info(explanation)

action\_explanation = generate\_action\_explanation(optimal\_action, hand\_strength, pot\_odds)

st.info(action\_explanation)

with tab4:

st.header("📊 Learning Analytics")

if st.session\_state.progress\_history:

df = pd.DataFrame(st.session\_state.progress\_history)

# Progress chart

fig = px.line(df, x='hand', y='strength',

title='Hand Strength Over Time',

labels={'strength': 'Hand Strength', 'hand': 'Hand Number'})

st.plotly\_chart(fig, use\_container\_width=True)

# Hand type distribution

hand\_counts = df['hand\_type'].value\_counts()

fig2 = px.bar(x=hand\_counts.index, y=hand\_counts.values,

title='Hand Types Analyzed',

labels={'x': 'Hand Type', 'y': 'Count'})

st.plotly\_chart(fig2, use\_container\_width=True)

# Summary stats

col1, col2, col3 = st.columns(3)

with col1:

st.metric("Total Hands", len(df))

with col2:

avg\_strength = df['strength'].mean()

st.metric("Avg Hand Strength", f"{avg\_strength:.1%}")

with col3:

accuracy = (st.session\_state.correct\_predictions / max(1, st.session\_state.hands\_analyzed)) \* 100

st.metric("Quiz Accuracy", f"{accuracy:.1f}%")

else:

st.info("📊 Analyze some hands to see your progress!")

# Footer

st.markdown("---")

st.markdown("""

\*\*🎯 Features:\*\*

- Real-time hand analysis and classification

- Action prediction based on pot odds and hand strength

- Adaptive tutorial system with skill-based explanations

- Interactive practice scenarios and quizzes

- Progress tracking and learning analytics

\*This poker AI uses rule-based algorithms optimized for educational purposes.\*

""")