# -\*- coding: utf-8 -\*-

import pandas as pd

import openpyxl

import streamlit as st

import requests

import matplotlib.pyplot as plt

from fpdf import FPDF

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error

# Streamlit Web App Title

st.title("AI-Powered Appraisal Adjustment Tool")

# Upload MLS Data File

uploaded\_file = st.file\_uploader("Upload MLS Data (CSV or Excel)", type=["csv", "xlsx"])

if uploaded\_file:

# Load Data

file\_extension = uploaded\_file.name.split(".")[-1]

if file\_extension == "csv":

df = pd.read\_csv(uploaded\_file)

else:

df = pd.read\_excel(uploaded\_file)

# Display Uploaded Data

st.write("Uploaded MLS Data:")

st.dataframe(df.head())

# Ensure Data Cleaning

df['Total Bedrooms'] = pd.to\_numeric(df['Total Bedrooms'], errors='coerce')

df['Total SqFt.'] = pd.to\_numeric(df['Total SqFt.'], errors='coerce')

df['Total Bathrooms'] = pd.to\_numeric(df['Total Bathrooms'], errors='coerce')

df['Garage Stall'] = pd.to\_numeric(df['Garage Stall'], errors='coerce')

df['Lot Acres'] = pd.to\_numeric(df['Lot Acres'], errors='coerce')

df['Sold Price'] = pd.to\_numeric(df['Sold Price'], errors='coerce')

df\_clean = df.dropna()

# Define Features and Target

features = ['Total Bedrooms', 'Total SqFt.', 'Total Bathrooms', 'Garage Stall', 'Lot Acres']

X = df\_clean[features]

y = df\_clean['Sold Price']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Models

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

xgb\_model = XGBRegressor(n\_estimators=100, learning\_rate=0.1, random\_state=42)

lr\_model = LinearRegression()

rf\_model.fit(X\_train, y\_train)

xgb\_model.fit(X\_train, y\_train)

lr\_model.fit(X\_train, y\_train)

# Predict and Average Results

rf\_pred = rf\_model.predict(X\_test)

xgb\_pred = xgb\_model.predict(X\_test)

lr\_pred = lr\_model.predict(X\_test)

final\_pred = (rf\_pred + xgb\_pred + lr\_pred) / 3

# Calculate Model Performance

final\_mae = mean\_absolute\_error(y\_test, final\_pred)

st.write(f"Model Accuracy (MAE): ${final\_mae:,.2f}")

# Integrate Real-Time Market Data from Zillow API

st.write("### Real-Time Market Data")

api\_url = "https://api.zillow.com/property-data"

params = {"zipcode": "83814", "property\_type": "single\_family"}

headers = {"Authorization": "Bearer YOUR\_ZILLOW\_API\_KEY"}

response = requests.get(api\_url, params=params, headers=headers)

if response.status\_code == 200:

market\_data = response.json()

st.write(market\_data)

else:

st.write("Failed to fetch market data from Zillow API.")

# Calculate Feature Importance

feature\_importance = dict(zip(features, rf\_model.feature\_importances\_))

adjustments = {

"Feature": features,

"Monetary Adjustment ($)": [feature\_importance[feature] \* (y.max() - y.min()) for feature in features],

"Percentage Adjustment (%)": [(feature\_importance[feature] \* (y.max() - y.min())) / y.median() \* 100 for feature in features]

}

df\_adjustments = pd.DataFrame(adjustments)

# Display Adjustments

st.write("### Calculated Appraisal Adjustments:")

st.dataframe(df\_adjustments)

# Plot Feature Importance

fig, ax = plt.subplots()

ax.barh(df\_adjustments['Feature'], df\_adjustments['Monetary Adjustment ($)'])

ax.set\_xlabel("Monetary Adjustment ($)")

ax.set\_title("Feature Importance in Appraisal Adjustments")

st.pyplot(fig)

# Save to Excel

excel\_file\_path = "appraisal\_adjustments.xlsx"

df\_adjustments.to\_excel(excel\_file\_path, index=False, engine='openpyxl')

st.download\_button(label="Download Adjustments Report (Excel)", data=open(excel\_file\_path, "rb").read(), file\_name=excel\_file\_path, mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")

# Generate PDF Report

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, txt="AI-Powered Real Estate Appraisal Report", ln=True, align="C")

pdf.cell(200, 10, txt=f"Model Accuracy (MAE): ${final\_mae:,.2f}", ln=True, align="L")

pdf.cell(200, 10, txt="Calculated Adjustments:", ln=True, align="L")

for index, row in df\_adjustments.iterrows():

pdf.cell(200, 10, txt=f"{row['Feature']}: ${row['Monetary Adjustment ($)']:,.2f}", ln=True, align="L")

pdf\_file\_path = "appraisal\_report.pdf"

pdf.output(pdf\_file\_path)

st.download\_button(label="Download Appraisal Report (PDF)", data=open(pdf\_file\_path, "rb").read(), file\_name=pdf\_file\_path, mime="application/pdf")