import streamlit as st

import pandas as pd

import seaborn as sns

import plotly.express as px

import matplotlib.pyplot as plt

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

from sklearn.preprocessing import StandardScaler, LabelEncoder

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

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.svm import SVR, SVC

from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

import pickle

# Title and description

st.title("Machine Learning Application")

st.write("This application allows you to perform machine learning tasks such as data loading, preprocessing, model selection, training, evaluation, and prediction.")

# Greetings button

if st.button("Greetings"):

    st.balloons()

    st.write("Hello there! Welcome to the Machine Learning Application.")

# Data loading

st.sidebar.header("Data Loading")

option = st.sidebar.radio("Select Data Source:", ("Upload Data", "Use Example Data"))

if option == "Upload Data":

    uploaded\_file = st.sidebar.file\_uploader("Upload Dataset", type=["csv", "xlsx", "tsv"])

    if uploaded\_file is not None:

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

else:

    default\_dataset = st.sidebar.selectbox("Select Default Dataset:", ("Titanic", "Tips", "Iris"))

    if default\_dataset == "Titanic":

        data = sns.load\_dataset("titanic")

    elif default\_dataset == "Tips":

        data = sns.load\_dataset("tips")

    else:

        data = sns.load\_dataset("iris")

# Basic information about the dataset

st.write("Columns:", data.columns)

st.write("Shape:", data.shape)

st.write("Number of Rows:", data.shape[0])

st.write("Number of columns:", data.shape[1])

st.write("Info:", data.dtypes)

st.write("Summary Statistics:", data.describe())

# Scale the Numeric columns having the float datatype

numerical\_columns=data.select\_dtypes(include=['float']).columns

# Lets Apply the Standard Scaler

scaler=StandardScaler()

data[numerical\_columns]=scaler.fit\_transform(data[numerical\_columns])

# Encoding categorical variables

for col in data.columns:

    if data[col].dtype == 'object' or data[col].dtype.name == 'category':

        le = LabelEncoder()

        data[col] = le.fit\_transform(data[col])

# Feature and target selection

selected\_features = st.sidebar.multiselect("Select Features:", data.columns)

selected\_target = st.sidebar.selectbox("Select Target Variable:", data.columns)

# Determine problem type

problem\_type = st.sidebar.radio("Select Problem Type:", ("Classification", "Regression"))

if problem\_type == "Classification":

    st.write("Problem Type: Classification")

else:

    st.write("Problem Type: Regression")

# Data preprocessing

if st.checkbox("Data Preprocessing"):

    # Handling missing values

    missing\_values\_percentage = data.isnull().sum() / len(data) \* 100

    columns\_to\_drop = missing\_values\_percentage[missing\_values\_percentage > 90].index

    data.drop(columns=columns\_to\_drop, inplace=True)

    if len(data.columns) > 0:

        imputer = IterativeImputer(max\_iter=50)

        data[selected\_features] = imputer.fit\_transform(data[selected\_features])

    # Feature scaling

    # scaler = StandardScaler()

    # data[selected\_features] = scaler.fit\_transform(data[selected\_features])

# Train test split

test\_size = st.sidebar.slider("Select Train Test Split Size:", 0.1, 0.9, 0.2)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[selected\_features], data[selected\_target], test\_size=test\_size, random\_state=42)

# Model selection

if st.checkbox("Model Selection"):

    if problem\_type == "Regression":

        model\_name = st.sidebar.selectbox("Select Regression Model:", ("Linear Regression", "SVR", "Decision Tree", "Random Forest"))

    else:

        model\_name = st.sidebar.selectbox("Select Classification Model:", ("Support Vector Classifier", "Logistic Classifier", "Decision Tree Classifier", "Random Forest Classifier"))

# Model training and evaluation

if st.button("Train Model"):

    if problem\_type == "Regression":

        if model\_name == "Linear Regression":

            model = LinearRegression()

        elif model\_name == "SVR":

            model = SVR()

        elif model\_name == "Decision Tree":

            model = DecisionTreeRegressor()

        else:

            model = RandomForestRegressor()

        model.fit(X\_train, y\_train)

        y\_pred = model.predict(X\_test)

        # Evaluation metrics

        st.write("Evaluation Metrics:")

        st.write("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

        st.write("Root Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred, squared=False))

        st.write("Mean Absolute Error:", mean\_absolute\_error(y\_test, y\_pred))

        st.write("R2 Score:", r2\_score(y\_test, y\_pred))

    else:

        if model\_name == "Support Vector Classifier":

            model = SVC()

        elif model\_name == "Logistic Classifier":

            model = LogisticRegression()

        elif model\_name == "Decision Tree Classifier":

            model = DecisionTreeClassifier()

        else:

            model = RandomForestClassifier()

        model.fit(X\_train, y\_train)

        y\_pred = model.predict(X\_test)

        # Evaluation metrics

        st.write("Evaluation Metrics:")

        st.write("Accuracy:", accuracy\_score(y\_test, y\_pred))

        st.write("Precision:", precision\_score(y\_test, y\_pred, average='weighted'))

        st.write("Recall:", recall\_score(y\_test, y\_pred, average='weighted'))

        st.write("F1 Score:", f1\_score(y\_test, y\_pred, average='weighted'))

        st.write("Confusion Matrix:")

        st.write(confusion\_matrix(y\_test, y\_pred))

# Automatic Exploratory Data Analysis

if st.checkbox("Automatic EDA"):

    st.write("Automatic EDA Results:")

    fig, axes = plt.subplots(nrows=1, ncols=len(numerical\_columns), figsize=(20, 5))

    for i, col in enumerate(numerical\_columns):

        sns.histplot(data[col], ax=axes[i], kde=True)

        axes[i].set\_title(col)

    st.pyplot(fig)

    fig = px.scatter\_matrix(data, dimensions=numerical\_columns)

    st.plotly\_chart(fig)