import streamlit as st

st.set\_page\_config(page\_title="Student Performance Prediction App", page\_icon="📊")

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.metrics import accuracy\_score, mean\_squared\_error

import numpy as np

# Load external stylesheet

with open("style.css") as f:

st.markdown(f"<style>{f.read()}</style>", unsafe\_allow\_html=True)

# Load data

data = pd.read\_csv('StudentPerformanceFactors.csv')

# Add binary target column for pass/fail

pass\_threshold = 70 # Define a passing threshold for exam scores

data['Pass\_Fail'] = data['Exam\_Score'].apply(lambda x: 1 if x >= pass\_threshold else 0)

# Sidebar for page selection

page = st.sidebar.selectbox("Select Page", ["Home", "Data Visualization", "Prediction"])

# Home Page

if page == "Home":

st.title("Welcome to the Student Performance Analysis App")

st.image('image\_home.jpeg', use\_column\_width=True)

st.write(

"""

This app predicts student performance based on various factors.

Use the Data Visualization page to explore insights in the dataset,

or head to the Prediction page to make predictions based on your inputs.

"""

)

# Data Visualization Page

elif page == "Data Visualization":

st.title("Data Visualization")

st.write("Explore various visualizations of the student performance dataset.")

# Sidebar for selecting plot type

plot\_type = st.sidebar.selectbox("Select Plot Type", ["Correlation Heatmap", "Bar Chart", "Pie Chart", "Line Chart", "Boxplot", "Scatter Plot", "Histogram"])

# Dropdown for feature selection based on the plot type

if plot\_type in ["Bar Chart", "Pie Chart", "Line Chart", "Boxplot", "Histogram"]:

feature = st.sidebar.selectbox("Select Feature", data.columns)

elif plot\_type == "Scatter Plot":

x\_feature = st.sidebar.selectbox("Select X-axis Feature", data.columns)

y\_feature = st.sidebar.selectbox("Select Y-axis Feature", data.columns)

# Plotting based on selected plot type

st.subheader(f"{plot\_type} Visualization")

# Correlation Heatmap

if plot\_type == "Correlation Heatmap":

st.write("Correlation Heatmap of the Dataset")

# Select only numerical columns for correlation calculation

numerical\_data = data.select\_dtypes(include=[np.number])

corr = numerical\_data.corr()

fig, ax = plt.subplots(figsize=(10, 8))

sns.heatmap(corr, annot=True, cmap='coolwarm', ax=ax)

st.pyplot(fig)

# Bar Chart

elif plot\_type == "Bar Chart":

st.write(f"Bar Chart of {feature}")

fig, ax = plt.subplots()

data[feature].value\_counts().plot(kind='bar', ax=ax)

ax.set\_xlabel(feature)

ax.set\_ylabel("Count")

st.pyplot(fig)

# Pie Chart

elif plot\_type == "Pie Chart":

st.write(f"Pie Chart of {feature}")

fig, ax = plt.subplots()

data[feature].value\_counts().plot(kind='pie', autopct='%1.1f%%', ax=ax)

ax.set\_ylabel('')

st.pyplot(fig)

# Line Chart

elif plot\_type == "Line Chart":

st.write(f"Line Chart of {feature}")

fig, ax = plt.subplots()

data[feature].plot(kind='line', ax=ax)

ax.set\_xlabel("Index")

ax.set\_ylabel(feature)

st.pyplot(fig)

# Boxplot

elif plot\_type == "Boxplot":

st.write(f"Boxplot of {feature}")

fig, ax = plt.subplots()

sns.boxplot(data=data, y=feature, ax=ax)

st.pyplot(fig)

# Scatter Plot

elif plot\_type == "Scatter Plot":

st.write(f"Scatter Plot between {x\_feature} and {y\_feature}")

fig, ax = plt.subplots()

sns.scatterplot(data=data, x=x\_feature, y=y\_feature, ax=ax)

st.pyplot(fig)

# Histogram

elif plot\_type == "Histogram":

st.write(f"Histogram of {feature}")

fig, ax = plt.subplots()

data[feature].plot(kind='hist', bins=20, ax=ax)

ax.set\_xlabel(feature)

st.pyplot(fig)

# Prediction Page

elif page == "Prediction":

st.title("Student Exam Performance Prediction")

# Define features and target variable for classification and regression

classification\_target = 'Pass\_Fail' # Binary target: 1 for pass, 0 for fail

regression\_target = 'Exam\_Score' # Target for score prediction

feature\_columns = ['Attendance', 'Hours\_Studied', 'Previous\_Scores', 'Access\_to\_Resources', 'Tutoring\_Sessions']

X = data[feature\_columns]

y\_class = data[classification\_target]

y\_reg = data[regression\_target]

# One-hot encode categorical features

X = pd.get\_dummies(X)

# Split data

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

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

# Train classification model

classifier = RandomForestClassifier()

classifier.fit(X\_train\_class, y\_train\_class)

# Calculate classification accuracy

y\_pred\_class = classifier.predict(X\_test\_class)

classification\_accuracy = accuracy\_score(y\_test\_class, y\_pred\_class)

st.write(f"Classification Model Accuracy: {classification\_accuracy \* 100:.2f}%")

# Train regression model

regressor = RandomForestRegressor()

regressor.fit(X\_train\_reg, y\_train\_reg)

# Calculate regression accuracy (RMSE)

y\_pred\_reg = regressor.predict(X\_test\_reg)

mse = mean\_squared\_error(y\_test\_reg, y\_pred\_reg)

rmse = np.sqrt(mse)

st.write(f"Regression Model RMSE: {rmse:.2f}")

# Input fields for the specific features

st.subheader("Enter student data:")

attendance = st.number\_input("Attendance (%)", min\_value=0.0, max\_value=100.0, step=1.0)

hours\_studied = st.number\_input("Hours Studied per Week", min\_value=0.0, max\_value=50.0, step=0.5)

previous\_scores = st.number\_input("Previous Scores (out of 100)", min\_value=0.0, max\_value=100.0, step=1.0)

access\_to\_resources = st.selectbox("Access to Resources", ['Low', 'Medium', 'High'])

tutoring\_sessions = st.number\_input("Number of Tutoring Sessions per Week", min\_value=0, max\_value=7, step=1)

# Prepare input data for prediction

input\_data = pd.DataFrame({

'Attendance': [attendance],

'Hours\_Studied': [hours\_studied],

'Previous\_Scores': [previous\_scores],

'Access\_to\_Resources': [access\_to\_resources],

'Tutoring\_Sessions': [tutoring\_sessions]

})

# One-hot encode input data and align with training data columns

input\_data = pd.get\_dummies(input\_data)

input\_data = input\_data.reindex(columns=X\_train\_class.columns, fill\_value=0)

# Make predictions

if st.button("Predict"):

# Predict pass/fail outcome

predicted\_class = classifier.predict(input\_data)[0]

grade = "🎉 Pass 🎉" if predicted\_class == 1 else "❌ Fail ❌"

# Predict exam score

predicted\_score = regressor.predict(input\_data)[0]

st.write(f"Predicted Outcome: {grade}")

st.write(f"Predicted Exam Score: {predicted\_score:.2f}")

# Display GIF based on pass/fail outcome

if predicted\_class == 1:

st.image('pass.gif', width=300) # Replace with your pass GIF file path

else:

st.image('fail.gif', width=300) # Replace with your fail GIF file path