# Air Quality Prediction Dashboard (Optimized Version)

import streamlit as st

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.metrics import mean\_squared\_error, r2\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Page configuration

st.set\_page\_config(

page\_title="🌫 Air Quality Prediction Dashboard",

page\_icon="🌍",

layout="wide",

initial\_sidebar\_state="expanded"

)

# Custom CSS

st.markdown("""

<style>

.main { background-color: #f8f9fa; }

.stButton>button { background-color: #4CAF50; color: white; }

.stSelectbox>div>div>select { background-color: #e9f7ef; }

.css-1aumxhk { background-color: #e9f7ef; }

</style>

""", unsafe\_allow\_html=True)

@st.cache\_data(show\_spinner=True)

def load\_data(uploaded\_file):

try:

df = pd.read\_csv(uploaded\_file, sep=';', low\_memory=False)

df = df.drop(columns=["Unnamed: 15", "Unnamed: 16"], errors='ignore')

for col in df.columns:

if col not in ["Date", "Time"]:

df[col] = df[col].astype(str).str.replace(',', '.', regex=False)

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

df.replace(-200, np.nan, inplace=True)

df.dropna(inplace=True)

try:

df['DateTime'] = pd.to\_datetime(df['Date'] + ' ' + df['Time'], format='%d/%m/%Y %H.%M.%S')

except ValueError:

try:

df['DateTime'] = pd.to\_datetime(df['Date'] + ' ' + df['Time'], format='%m/%d/%Y %H.%M.%S')

except ValueError:

st.warning("Automatic datetime parsing used")

df['DateTime'] = pd.to\_datetime(df['Date'] + ' ' + df['Time'], errors='coerce')

df.dropna(subset=['DateTime'], inplace=True)

return df

except Exception as e:

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

return None

def model\_hyperparameters():

st.sidebar.subheader("🔧 Model Parameters")

n\_estimators = st.sidebar.slider("Number of trees", 10, 200, 100, 10)

max\_depth = st.sidebar.slider("Max depth", 2, 20, 10, 1)

return n\_estimators, max\_depth

def get\_feature\_target(df, target\_col):

X = df.drop(columns=["Date", "Time", "DateTime", target\_col], errors='ignore')

y = df[target\_col]

return X, y

def show\_data\_summary(df):

st.subheader("📊 Data Overview")

col1, col2 = st.columns(2)

with col1:

st.write("\*Dataset Shape:\*", df.shape)

st.write("\*Columns:\*", list(df.columns))

with col2:

st.write("\*Missing Values:\*")

st.write(df.isnull().sum())

st.subheader("📈 Data Statistics")

st.write(df.describe())

st.subheader("⏳ CO(GT) Over Time")

fig, ax = plt.subplots(figsize=(12, 4))

df.set\_index('DateTime')['CO(GT)'].plot(ax=ax)

plt.title("CO Levels Over Time")

st.pyplot(fig)

st.subheader("🔥 Correlation Heatmap")

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

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

sns.heatmap(df[numeric\_cols].corr(), annot=True, cmap='coolwarm', ax=ax)

st.pyplot(fig)

def regression\_model(df):

st.subheader("🔢 Regression Model: Predict CO(GT)")

n\_estimators, max\_depth = model\_hyperparameters()

X, y = get\_feature\_target(df, "CO(GT)")

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

model = RandomForestRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42)

model.fit(X\_train, y\_train)

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

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

r2 = r2\_score(y\_test, y\_pred)

col1, col2 = st.columns(2)

with col1:

st.metric("Mean Squared Error", f"{mse:.4f}")

with col2:

st.metric("R² Score", f"{r2:.4f}")

st.subheader("📊 Feature Importance")

importance = pd.Series(model.feature\_importances\_, index=X.columns).sort\_values()

fig, ax = plt.subplots(figsize=(10, 6))

sns.barplot(x=importance.values, y=importance.index, palette="viridis", ax=ax)

st.pyplot(fig)

st.subheader("🔄 Actual vs Predicted Values")

fig, ax = plt.subplots(figsize=(8, 6))

sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.6, ax=ax)

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')

plt.xlabel("Actual")

plt.ylabel("Predicted")

st.pyplot(fig)

def classification\_model(df):

st.subheader("🏷 Classification Model: Air Quality Labels")

n\_estimators, max\_depth = model\_hyperparameters()

def label\_air\_quality(co):

if co <= 2:

return 'Good'

elif co <= 5:

return 'Moderate'

else:

return 'Bad'

df['AirQuality'] = df['CO(GT)'].apply(label\_air\_quality)

st.write("\*Class Distribution:\*")

dist = df['AirQuality'].value\_counts(normalize=True)

st.write(dist)

fig, ax = plt.subplots(figsize=(6, 4))

dist.plot(kind='bar', color=['green', 'orange', 'red'], ax=ax)

plt.title("Air Quality Class Distribution")

st.pyplot(fig)

X, y = get\_feature\_target(df, "AirQuality")

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

model = RandomForestClassifier(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=42, class\_weight='balanced')

model.fit(X\_train, y\_train)

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

st.subheader("📝 Classification Report")

st.table(pd.DataFrame(classification\_report(y\_test, y\_pred, output\_dict=True)).transpose())

st.subheader("🔄 Confusion Matrix")

cm = confusion\_matrix(y\_test, y\_pred, labels=['Good', 'Moderate', 'Bad'])

fig, ax = plt.subplots(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Good', 'Moderate', 'Bad'], yticklabels=['Good', 'Moderate', 'Bad'], ax=ax)

ax.set\_xlabel("Predicted")

ax.set\_ylabel("Actual")

st.pyplot(fig)

def main():

st.title("🌫 Air Quality Prediction Dashboard")

with st.expander("ℹ How to use this dashboard"):

st.write("""

1. Upload your AirQuality.csv file (semicolon-separated)

2. Select the model type

3. Adjust model parameters

4. View the results and download cleaned data

""")

sample = """Date;Time;CO(GT);PT08.S1(CO);NMHC(GT);C6H6(GT);PT08.S2(NMHC);NOx(GT);PT08.S3(NOx);NO2(GT);PT08.S4(NO2);PT08.S5(O3);T;RH;AH\n01/01/2004;12.00.00;2,6;1360;150;11,9;1046;166;1056;113;1692;1268;13,6;48,9;0,7578"""

st.download\_button("Download sample data format", data=sample.encode('utf-8'), file\_name="AirQuality\_sample.csv", mime="text/csv")

uploaded\_file = st.file\_uploader("Upload your AirQuality.csv file", type=["csv"])

if uploaded\_file:

df = load\_data(uploaded\_file)

if df is not None:

mode = st.sidebar.selectbox("Choose a model type", ["Data Exploration", "Regression (Predict CO)", "Classification (Air Quality)"])

if mode == "Data Exploration":

show\_data\_summary(df)

elif mode == "Regression (Predict CO)":

regression\_model(df)

elif mode == "Classification (Air Quality)":

classification\_model(df)

csv = df.to\_csv(index=False, sep=';').encode('utf-8')

st.sidebar.download\_button("Download Cleaned Data", data=csv, file\_name='cleaned\_air\_quality.csv', mime='text/csv')

if \_name\_ == "\_main\_":

main()