Name : Shivam Bhatt

Enrollment No: 92301733046

Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression, Lasso, Ridge

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.impute import SimpleImputer # For handling potential missing values

# --- Configuration ---

FILE\_PATH\_METHANE = '/content/ethylene\_methane.txt'

FILE\_PATH\_CO = '/content/ethylene\_CO.txt'

# --- 1. Create Mock Data (Replace with your actual file loading) ---

# Assuming 'Ethylene' is the target, 'Methane'/'CO' are key features, plus others.

# Let's simulate some relationships.

# Mock Data for ethylene\_methane.txt

np.random.seed(42)

num\_samples = 100

data\_methane = {

    'Methane\_Concentration': np.random.rand(num\_samples) \* 100 + 50, # 50-150 ppm

    'Temperature\_C': np.random.rand(num\_samples) \* 30 + 10,       # 10-40 C

    'Humidity\_Percent': np.random.rand(num\_samples) \* 50 + 30,    # 30-80 %

    'Pressure\_kPa': np.random.rand(num\_samples) \* 20 + 90,        # 90-110 kPa

    'Ethylene\_Concentration': (

        5 + 0.8 \* (np.random.rand(num\_samples) \* 100 + 50)  # Methane effect

        + 0.5 \* (np.random.rand(num\_samples) \* 30 + 10)  # Temperature effect

        - 0.2 \* (np.random.rand(num\_samples) \* 50 + 30)  # Humidity effect

        + np.random.randn(num\_samples) \* 5 # Noise

    )

}

df\_methane = pd.DataFrame(data\_methane)

# Introduce some missing values for demonstration

df\_methane.loc[df\_methane.sample(frac=0.05).index, 'Methane\_Concentration'] = np.nan

df\_methane.loc[df\_methane.sample(frac=0.03).index, 'Temperature\_C'] = np.nan

print("--- Mock df\_methane Head ---")

print(df\_methane.head())

print("\n--- Mock df\_methane Info ---")

df\_methane.info()

print("\n")

# Mock Data for ethylene\_CO.txt

data\_co = {

    'CO\_Concentration': np.random.rand(num\_samples) \* 20 + 1,    # 1-21 ppm

    'Light\_Intensity\_Lux': np.random.rand(num\_samples) \* 1000 + 100, # 100-1100 Lux

    'Flow\_Rate\_LPM': np.random.rand(num\_samples) \* 5 + 1,         # 1-6 LPM

    'Ethylene\_Concentration': (

        2 + 1.2 \* (np.random.rand(num\_samples) \* 20 + 1)  # CO effect

        + 0.01 \* (np.random.rand(num\_samples) \* 1000 + 100) # Light effect

        + np.random.randn(num\_samples) \* 3 # Noise

    )

}

df\_co = pd.DataFrame(data\_co)

# Introduce some missing values for demonstration

df\_co.loc[df\_co.sample(frac=0.07).index, 'CO\_Concentration'] = np.nan

print("--- Mock df\_co Head ---")

print(df\_co.head())

print("\n--- Mock df\_co Info ---")

df\_co.info()

print("\n")

# --- Function to process and model each dataset ---

def analyze\_dataset(df, dataset\_name, target\_column='Ethylene\_Concentration'):

    print(f"\n--- Analyzing Dataset: {dataset\_name} ---")

    # --- 1. Data Understanding & Preprocessing ---

    print("\n1. Data Understanding & Preprocessing")

    print("\nInitial Data Info:")

    df.info()

    print("\nInitial Data Description:")

    print(df.describe())

    print("\nMissing Values Before Imputation:")

    print(df.isnull().sum())

    # Drop rows where the target variable is missing (critical for training)

    df.dropna(subset=[target\_column], inplace=True)

    # Impute missing values for features (using mean for numerical)

    # This imputer will be fit on training data and transform train/test

    imputer = SimpleImputer(strategy='mean')

    # Identify numerical columns for imputation, excluding the target if it's already handled

    numerical\_cols = df.select\_dtypes(include=np.number).columns.tolist()

    if target\_column in numerical\_cols:

        numerical\_cols.remove(target\_column)

    # Apply imputer to the numerical columns

    df[numerical\_cols] = imputer.fit\_transform(df[numerical\_cols])

    print("\nMissing Values After Imputation:")

    print(df.isnull().sum())

    # Visualize distributions

    plt.figure(figsize=(15, 5))

    for i, col in enumerate(df.drop(columns=[target\_column]).columns):

        plt.subplot(1, len(df.drop(columns=[target\_column]).columns), i + 1)

        sns.histplot(df[col], kde=True)

        plt.title(f'Distribution of {col}')

    plt.tight\_layout()

    plt.show()

    plt.figure(figsize=(8, 6))

    sns.histplot(df[target\_column], kde=True)

    plt.title(f'Distribution of {target\_column}')

    plt.show()

    # Correlation Matrix

    plt.figure(figsize=(10, 8))

    sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")

    plt.title(f'Correlation Matrix for {dataset\_name}')

    plt.show()

    # Define features (X) and target (y)

    X = df.drop(columns=[target\_column])

    y = df[target\_column]

    # Train-Test Split

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

    print(f"\nTrain set size: {X\_train.shape[0]} samples")

    print(f"Test set size: {X\_test.shape[0]} samples")

    # Feature Scaling (Standardization)

    scaler = StandardScaler()

    X\_train\_scaled = scaler.fit\_transform(X\_train)

    X\_test\_scaled = scaler.transform(X\_test)

    # Convert scaled arrays back to DataFrames for easier handling if needed (optional)

    X\_train\_scaled\_df = pd.DataFrame(X\_train\_scaled, columns=X\_train.columns, index=X\_train.index)

    X\_test\_scaled\_df = pd.DataFrame(X\_test\_scaled, columns=X\_test.columns, index=X\_test.index)

    # --- 2. Model Implementation & 3. Performance Evaluation ---

    print("\n2. Model Implementation & 3. Performance Evaluation")

    models = {

        'Linear Regression': LinearRegression(),

        'Lasso Regression': Lasso(alpha=0.1, random\_state=42), # Alpha is a hyperparameter

        'Ridge Regression': Ridge(alpha=1.0, random\_state=42), # Alpha is a hyperparameter

        'Decision Tree Regressor': DecisionTreeRegressor(random\_state=42, max\_depth=5), # max\_depth is a hyperparameter

        'Random Forest Regressor': RandomForestRegressor(n\_estimators=100, random\_state=42, max\_depth=8) # n\_estimators, max\_depth are hyperparameters

    }

    results = {}

    for name, model in models.items():

        print(f"\n--- Training {name} ---")

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

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

        mae = mean\_absolute\_error(y\_test, y\_pred)

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

        rmse = np.sqrt(mse)

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

        results[name] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R2': r2}

        print(f"{name} Performance:")

        print(f"  MAE: {mae:.4f}")

        print(f"  MSE: {mse:.4f}")

        print(f"  RMSE: {rmse:.4f}")

        print(f"  R-squared: {r2:.4f}")

        # Plot Actual vs. Predicted values

        plt.figure(figsize=(8, 6))

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

        plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2)

        plt.xlabel("Actual Values")

        plt.ylabel("Predicted Values")

        plt.title(f'{name}: Actual vs. Predicted ({dataset\_name})')

        plt.grid(True)

        plt.show()

        # Plot Residuals

        residuals = y\_test - y\_pred

        plt.figure(figsize=(8, 6))

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

        plt.axhline(y=0, color='r', linestyle='--')

        plt.xlabel("Predicted Values")

        plt.ylabel("Residuals")

        plt.title(f'{name}: Residual Plot ({dataset\_name})')

        plt.grid(True)

        plt.show()

    # --- 4. Analysis & Insights ---

    print("\n4. Analysis & Insights")

    results\_df = pd.DataFrame(results).T

    print("\n--- All Model Performance Summary ---")

    print(results\_df.sort\_values(by='R2', ascending=False))

    best\_model\_name = results\_df['R2'].idxmax()

    print(f"\nBest performing model based on R-squared: {best\_model\_name}")

    # Feature Importance (for tree-based models)

    if best\_model\_name in ['Decision Tree Regressor', 'Random Forest Regressor']:

        print(f"\n--- Feature Importance for {best\_model\_name} ---")

        model = models[best\_model\_name]

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

        print(feature\_importances)

        plt.figure(figsize=(10, 6))

        sns.barplot(x=feature\_importances.values, y=feature\_importances.index)

        plt.title(f'Feature Importance for {best\_model\_name} ({dataset\_name})')

        plt.xlabel('Importance')

        plt.ylabel('Feature')

        plt.show()

    elif best\_model\_name in ['Linear Regression', 'Lasso Regression', 'Ridge Regression']:

        print(f"\n--- Coefficients for {best\_model\_name} ---")

        model = models[best\_model\_name]

        coefficients = pd.Series(model.coef\_, index=X.columns).sort\_values(ascending=False)

        print(coefficients)

        plt.figure(figsize=(10, 6))

        sns.barplot(x=coefficients.values, y=coefficients.index)

        plt.title(f'Feature Coefficients for {best\_model\_name} ({dataset\_name})')

        plt.xlabel('Coefficient Value')

        plt.ylabel('Feature')

        plt.show()

    print("\n--- General Insights ---")

    print(f"The {best\_model\_name} generally performed best for {dataset\_name} based on R-squared.")

    print("Consider further hyperparameter tuning (e.g., using GridSearchCV or RandomizedSearchCV) for each model to optimize performance.")

    print("The correlation matrix and feature importance plots give insights into which factors are most influential on Ethylene concentration.")

    print("Residual plots can indicate if the model is missing any patterns (e.g., non-linearity). A good residual plot should show no discernible pattern.")

    print(f"\n--- End of Analysis for {dataset\_name} ---")

# --- Run Analysis for each dataset ---

# Step 1: Replace mock data with actual file loading

# For ethylene\_methane.txt

try:

    # Assuming your file is space-separated, comma-separated, or tab-separated

    # Adjust `sep` parameter if your delimiter is different

    df\_methane\_actual = pd.read\_csv(FILE\_PATH\_METHANE, sep='\s+') # Example for space-separated

    print(f"\nSuccessfully loaded {FILE\_PATH\_METHANE}")

    analyze\_dataset(df\_methane\_actual.copy(), "Ethylene\_Methane Dataset") # Use .copy() to avoid modifying original df

except FileNotFoundError:

    print(f"\nWarning: {FILE\_PATH\_METHANE} not found. Using mock data for Ethylene\_Methane analysis.")

    analyze\_dataset(df\_methane.copy(), "Ethylene\_Methane Dataset")

except Exception as e:

    print(f"\nError loading {FILE\_PATH\_METHANE}: {e}. Using mock data for Ethylene\_Methane analysis.")

    analyze\_dataset(df\_methane.copy(), "Ethylene\_Methane Dataset")

# For ethylene\_CO.txt

try:

    df\_co\_actual = pd.read\_csv(FILE\_PATH\_CO, sep='\s+') # Example for space-separated

    print(f"\nSuccessfully loaded {FILE\_PATH\_CO}")

    analyze\_dataset(df\_co\_actual.copy(), "Ethylene\_CO Dataset")

except FileNotFoundError:

    print(f"\nWarning: {FILE\_PATH\_CO} not found. Using mock data for Ethylene\_CO analysis.")

    analyze\_dataset(df\_co.copy(), "Ethylene\_CO Dataset")

except Exception as e:

    print(f"\nError loading {FILE\_PATH\_CO}: {e}. Using mock data for Ethylene\_CO analysis.")

    analyze\_dataset(df\_co.copy(), "Ethylene\_CO Dataset")



































































