dsbda-b2

April 28, 2025

```
[ ]: 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 LabelEncoder
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
[ ]: data = pd.read_csv("airquality.csv")
     print("  Original Air Quality Data:\n", data.head())
```

a. Data Cleaning - Handling Missing Values & Duplicates

```
[ ]: print(data.isnull().sum())
```

```
[ ]: # Fill missing numeric values with mean
     data['Ozone'] = data['Ozone'].fillna(data['Ozone'].mean())
     data['Solar.R'] = data['Solar.R'].fillna(data['Solar.R'].mean())
     data['Temp'] = data['Temp'].fillna(data['Temp'].mean())
```

```
[ ]: # Fill missing categorical values in 'Humidity' using mode
     data['Humidity'] = data['Humidity'].fillna(data['Humidity'].mode()[0])
```

```
[ ]: # Convert categorical 'Humidity' to numeric
     encoder = LabelEncoder()
     data['Humidity'] = encoder.fit_transform(data['Humidity'])
```

```
[ ]: # Remove duplicate records
     data.drop_duplicates(inplace=True)
```

```
[ ]: print( data.head())
```

#b. Error Correcting - Handling Invalid Values

```
[ ]: # Removing invalid values
     data = data[data['Wind'] >= 0]   # Wind cannot be negative
     data = data[(data['Month'] >= 1) & (data['Month'] <= 12)]   # Month range check
```

```
data = data[(data['Day'] >= 1) & (data['Day'] <= 31)]  # Day range check
```

[ ]:
```
print("\n  Data After Error Correction:\n", data.head())
```

#c. Data Transformation - Outlier Handling

[ ]:
```
# List of numerical columns to check for outliers
num_cols = ['Ozone', 'Solar.R', 'Wind', 'Temp']
```

[ ]:
```
for col in num_cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Cap outliers
    data[col] = data[col].clip(lower_bound, upper_bound)
```

[ ]:
```
print( data.head())
```

#d. Data Model Building - Predicting Ozone **Levels**

[ ]:
```
# Define Features (X) and Target Variable (y)
X = data[['Solar.R', 'Wind', 'Temp', 'Humidity']]
y = data['Ozone']
```

[ ]:
```
# Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
print( X_train.shape, "X_test shape:", X_test.shape)
```

```
(122, 4) X_test shape: (31, 4)
```

[ ]:
```
# Train a Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
# Make Predictions
y_pred = model.predict(X_test)
```

[ ]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, color='blue', alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')
plt.xlabel("Actual Ozone Levels")
plt.ylabel("Predicted Ozone Levels")
plt.title("Actual vs Predicted Ozone Levels")
plt.show()
```

1 e. Model Evaluation

[ ]:
```
# Calculate errors
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)
```

[ ]:
```
print(f"\n Model Performance:")
print(f" Mean Absolute Error (MAE): {mae:.2f}")
print(f" Mean Squared Error (MSE): {mse:.2f}")
print(f" Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f" R² Score: {r2:.2f}")
```

[ ]:
```

```