Experiment 2.3

Student Name: Rohan Singh Samant **UID: 21BCS9687**
Branch: CSE **Section/Group: 646/B**
Semester: 5 **Date of Performance: 12/9/23**
Subject Name: AIML **Subject Code:21CSH-316**

Aim: Implementing Linear Regression and Logistic Regression models

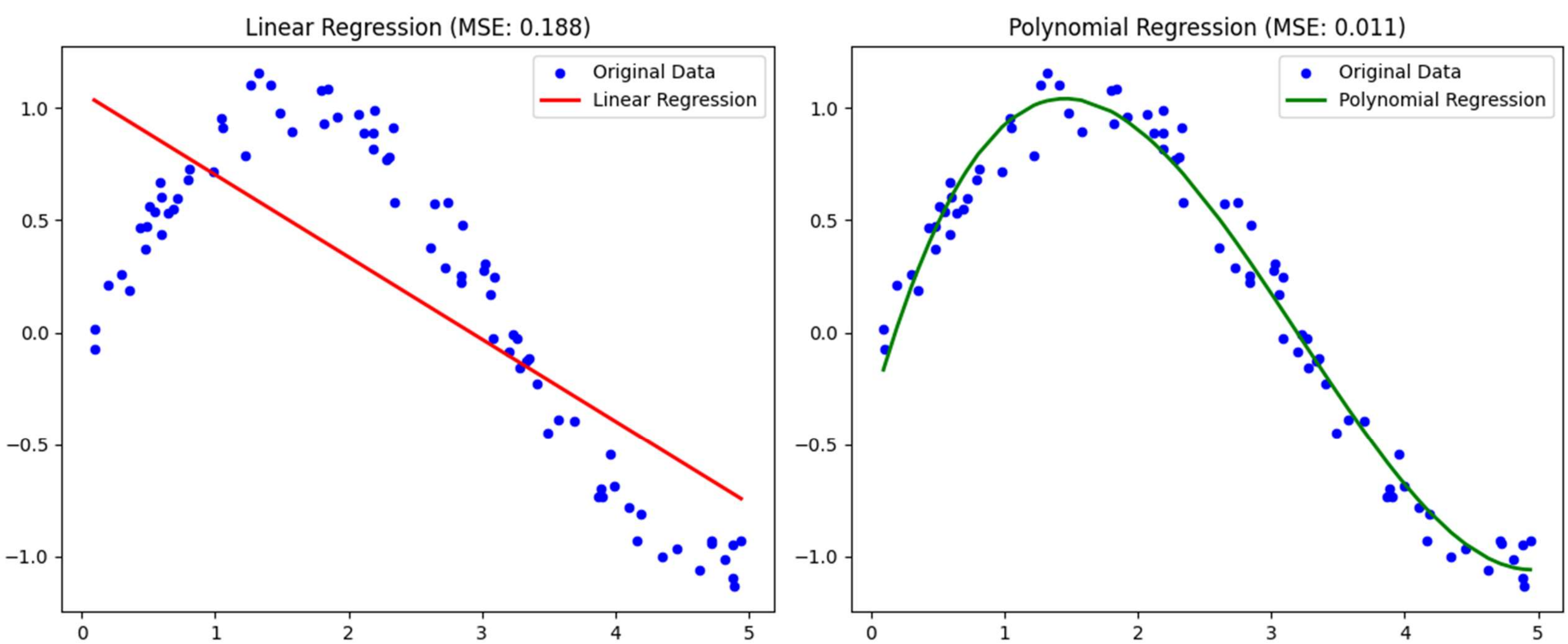
Objective: Your independent variables are highly correlated, causing instability in coefficient estimates. Solution: Use techniques like VIF (Variance Inflation Factor) or PCA (Principal Component Analysis) to identify and address multicollinearity

Program and output:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error


# Generate some synthetic data with a non-linear relationship

np.random.seed(0)

X = np.sort(5 * np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])


# Create a scatterplot of the original data
```

```
plt.figure(figsize=(12, 5))
plt.scatter(X, y, s=20, label="Original Data", color='blue')

# Linear regression model (Before Solution)
lr = LinearRegression()
lr.fit(X, y)
y_pred_lr = lr.predict(X)
mse_lr = mean_squared_error(y, y_pred_lr)

# Polynomial regression model (After Solution)
poly = PolynomialFeatures(degree=3)  # Adjust degree as needed
X_poly = poly.fit_transform(X)
lr_poly = LinearRegression()
lr_poly.fit(X_poly, y)
y_pred_poly = lr_poly.predict(X_poly)
mse_poly = mean_squared_error(y, y_pred_poly)

# Sort data points for smooth plotting
X_sorted = np.sort(X, axis=0)
y_pred_lr_sorted = lr.predict(X_sorted)
y_pred_poly_sorted = lr_poly.predict(poly.transform(X_sorted))

# Create plots for "Before" and "After" solutions
plt.subplot(1, 2, 1)
```

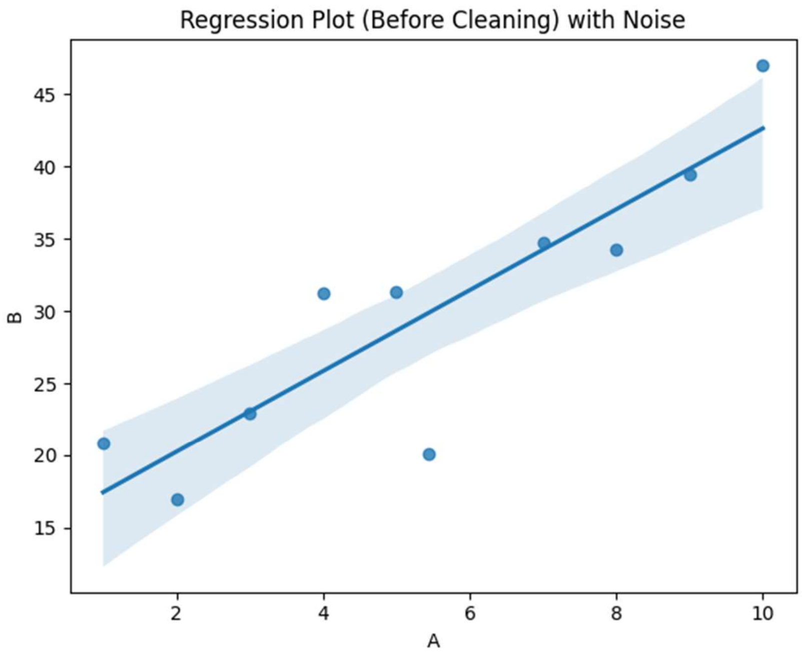
```
plt.scatter(X, y, s=20, label="Original Data", color='blue')
plt.plot(X_sorted, y_pred_lr_sorted, color='red', linewidth=2, label="Linear
Regression")
plt.title(f"Linear Regression (MSE: {mse_lr:.3f})")
plt.legend()


plt.subplot(1, 2, 2)
plt.scatter(X, y, s=20, label="Original Data", color='blue')
plt.plot(X_sorted, y_pred_poly_sorted, color='green', linewidth=2,
label="Polynomial Regression")
plt.title(f"Polynomial Regression (MSE: {mse_poly:.3f})")
plt.legend()


plt.tight_layout()
plt.show()
```

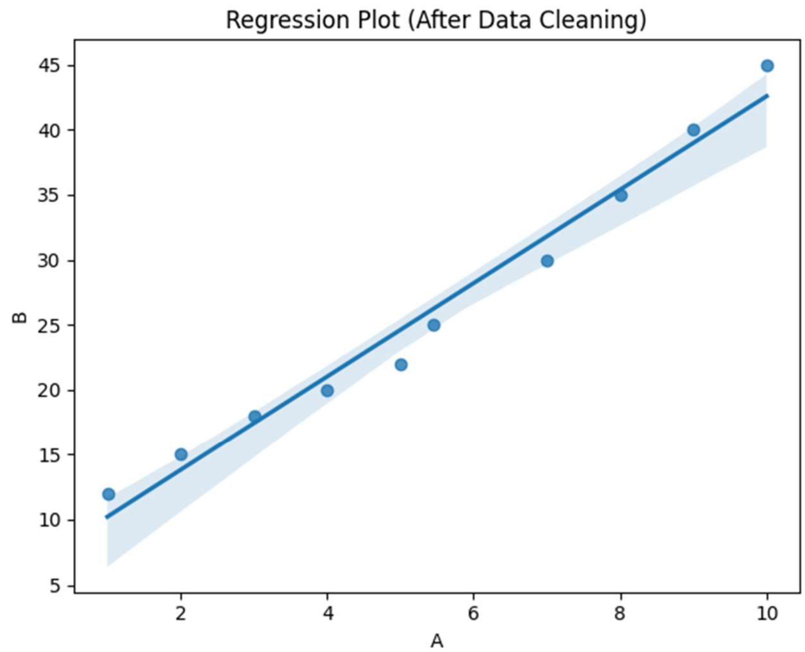


Problem: Your dataset contains missing values or outliers. Solution: Handle missing values through imputation or removal and address outliers using techniques like trimming or transformation.

```
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset with missing values and outliers
data = {
    'A': [1, 2, 3, 4, 5, np.nan, 7, 8, 9, 10],
    'B': [12, 15, 18, 20, 22, 25, 30, 35, 40, 45]
}

df = pd.DataFrame(data)

# Step 1: Handling missing values through imputation
mean_A = df['A'].mean()
df['A'].fillna(mean_A, inplace=True)

# Step 2: Addressing outliers using trimming
z_scores = np.abs(stats.zscore(df['B']))
threshold = 2
```

```
df = df[(z_scores < threshold)]

# Create regression plots before and after data cleaning
plt.figure(figsize=(12, 5))

# Before data cleaning with added noise
np.random.seed(0)
noise = np.random.normal(0, 5, len(df))  # Add random noise to the 'B' column
df_noisy = df.copy()
df_noisy['B'] += noise

plt.subplot(1, 2, 1)
sns.regplot(x='A', y='B', data=df_noisy)  # Use df_noisy for the "Before Cleaning" plot
plt.title('Regression Plot (Before Cleaning) with Noise')

# After data cleaning
plt.subplot(1, 2, 2)
sns.regplot(x='A', y='B', data=df)
plt.title('Regression Plot (After Data Cleaning)')

plt.tight_layout()
plt.show()
```


Regression Plot (Before Cleaning) with Noise
B
45
40
35
30
25
20
15
2
4
6
8
10
A


Regression Plot (After Data Cleaning)
B
45
40
35
30
25
20
15
10
5
2
4
6
8
10
A