**Program For Feature Engineering**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

# Step 3: Load dataset (example: Iris dataset)

iris = load\_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['target'] = iris.target

# Display the first few rows

df.head()

# Step 4: Feature Engineering Techniques

# 4.1: Handling Missing Values

# Simulating missing values in the 'petal length' column

df.loc[5:10, 'petal length (cm)'] = np.nan

# Using SimpleImputer to fill missing values with the median

imputer = SimpleImputer(strategy='median')

df['petal length (cm)'] = imputer.fit\_transform(df[['petal length (cm)']])

# 4.2: Feature Transformation (e.g., Log Transformation)

# Apply log transformation to the 'petal length' feature to reduce skewness

df['log\_petal\_length'] = np.log(df['petal length (cm)'] + 1)

# 4.3: Feature Scaling (Standardization)

# Standardize all the features (i.e., scale them to have mean=0 and std=1)

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(df[iris.feature\_names])

# Add scaled features back into the dataframe

df\_scaled = pd.DataFrame(scaled\_features, columns=iris.feature\_names)

df\_scaled['target'] = df['target']

# 4.4: Feature Creation (Interaction Features)

# Create an interaction feature: ratio between petal length and petal width

df['petal\_ratio'] = df['petal length (cm)'] / df['petal width (cm)']

# 4.5: Categorical Variable Encoding (One-Hot Encoding for target)

# If target was categorical, we'd one-hot encode it (but here, it's already numeric)

# df = pd.get\_dummies(df, columns=['target'], drop\_first=True)

# Step 5: Visualizing the Feature Engineering Results

# Plotting distributions of original vs transformed features

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

sns.histplot(df['petal length (cm)'], kde=True, ax=axes[0, 0], color='blue')

axes[0, 0].set\_title('Original Petal Length')

sns.histplot(df['log\_petal\_length'], kde=True, ax=axes[0, 1], color='red')

axes[0, 1].set\_title('Log Transformed Petal Length')

sns.boxplot(data=df, x='target', y='petal length (cm)', ax=axes[1, 0])

axes[1, 0].set\_title('Boxplot: Petal Length by Target Class')

sns.scatterplot(x=df['petal length (cm)'], y=df['petal width (cm)'], hue=df['target'], ax=axes[1, 1])

axes[1, 1].set\_title('Petal Length vs Petal Width')

plt.tight\_layout()

plt.show()

# Step 6: Train/Test Split

# Split the data into features and target variable

X = df.drop(columns=['target'])

y = df['target']

# Split the dataset into training and testing sets (80/20 split)

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

# Check the shapes of the resulting sets

print(f"Training feature set shape: {X\_train.shape}")

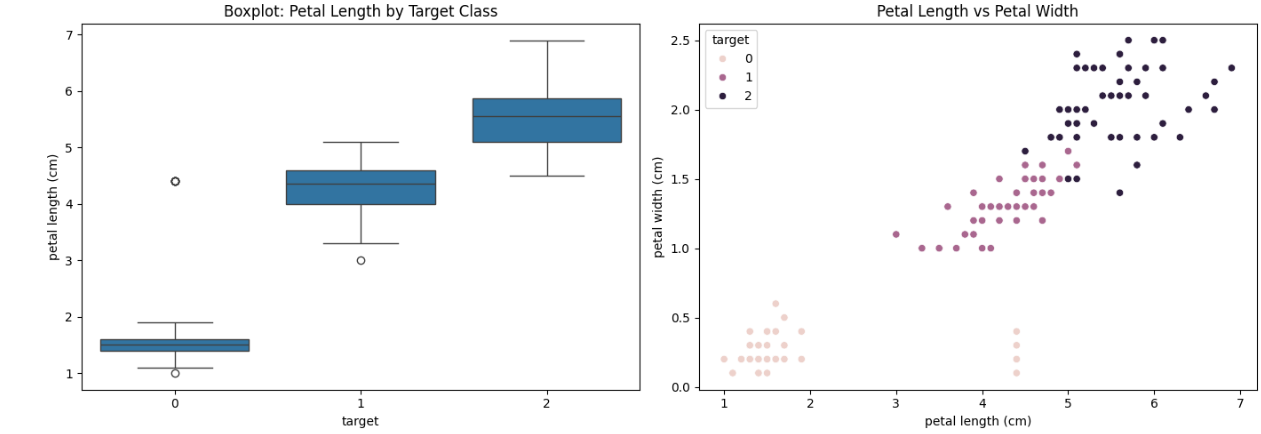
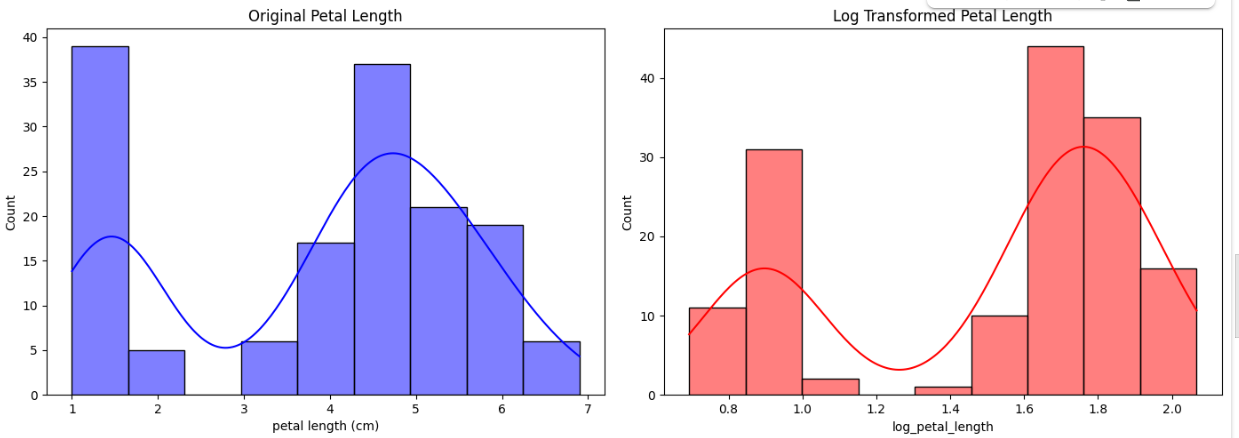
print(f"Testing feature set shape: {X\_test.shape}")

# Step 7: Output the feature-engineered DataFrame

df.head()

# Step 8: Conclusion Message

print("Feature engineering steps are completed, and data is ready for modeling!")

OUTPUT :