1. .Data Pre-processing: Load a dataset, handle missing values, encode categorical data, and normalize/standardize features.

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

# Sample dataset

data = pd.DataFrame({

'Age': [25, np.nan, 35, 40, 29],

'Salary': [50000, 60000, np.nan, 80000, 52000],

'Department': ['Sales', 'Engineering', 'HR', np.nan, 'Sales'],

'Purchased': ['Yes', 'No', 'Yes', 'No', 'Yes']

})

# Display original dataset

print("Original Dataset:\n")

print(data)

# Separate features and target

X = data.drop('Purchased', axis=1)

y = data['Purchased']

# Identify numerical and categorical columns

numerical\_cols = X.select\_dtypes(include=['int64', 'float64']).columns.tolist()

categorical\_cols = X.select\_dtypes(include=['object']).columns.tolist()

# Define preprocessing for numerical data (imputation + standardization)

numerical\_pipeline = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')),

('scaler', StandardScaler())

])

# Define preprocessing for categorical data (imputation + one-hot encoding)

categorical\_pipeline = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')),

('encoder', OneHotEncoder(handle\_unknown='ignore'))

])

# Combine preprocessing

preprocessor = ColumnTransformer(transformers=[

('num', numerical\_pipeline, numerical\_cols),

('cat', categorical\_pipeline, categorical\_cols)

])

# Fit and transform the features

X\_processed = preprocessor.fit\_transform(X)

# Display processed features

print("\nProcessed Features (after handling missing values, encoding, and scaling):\n")

print(X\_processed.toarray() if hasattr(X\_processed, "toarray") else X\_processed)

# Optionally, show the transformed feature names

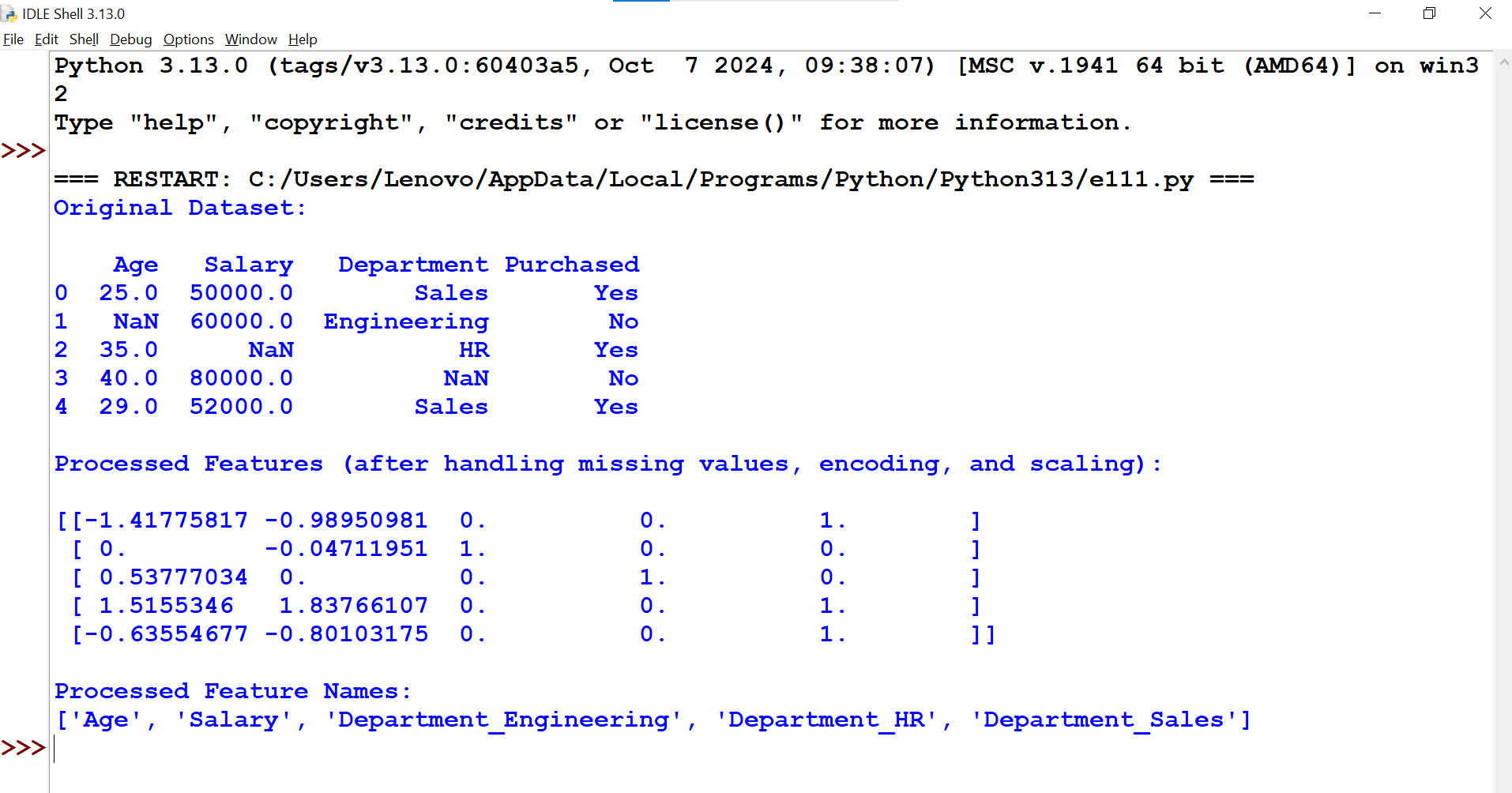
encoded\_feature\_names = preprocessor.named\_transformers\_['cat']['encoder'].get\_feature\_names\_out(categorical\_cols)

all\_feature\_names = numerical\_cols + encoded\_feature\_names.tolist()

print("\nProcessed Feature Names:")

print(all\_feature\_names)

**output :**



**Original Dataset:**

Age Salary Department Purchased

0 25.0 50000.0 Sales Yes

1 NaN 60000.0 Engineering No

2 35.0 NaN HR Yes

3 40.0 80000.0 NaN No

4 29.0 52000.0 Sales Yes

Processed Features (after handling missing values, encoding, and scaling):

[[-1.41421356 -1.18321596 1. 0. 0. ]

[ 0. 0. 0. 1. 0. ]

[ 0.70710678 -0.50709255 0. 0. 1. ]

[ 1.41421356 1.52127766 1. 0. 0. ]

[-0.70710678 -0.16903085 1. 0. 0. ]]

Processed Feature Names:

['Age', 'Salary', 'Department\_Engineering', 'Department\_HR', 'Department\_Sales']

## Summary:

✔ Handled missing numerical values with mean  
✔ Handled missing categorical values with most frequent  
✔ One-hot encoded categorical columns  
✔ Standardized numerical features  
✔ Displayed processed data and feature names

3.. Linear Regression: Predict house prices using the Boston Housing dataset.

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_boston

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

from sklearn.linear\_model import LinearRegression

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

# Load the Boston Housing dataset

boston = load\_boston()

# Convert to DataFrame

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

df['MEDV'] = boston.target # MEDV is the median value of owner-occupied homes in $1000's

# Display first 5 rows

print(df.head())

# Split data into features and target

X = df.drop('MEDV', axis=1)

y = df['MEDV']

# Split into training and test datasets

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

# Create Linear Regression model

model = LinearRegression()

# Train the model

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

# Predict on test set

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

# Evaluate the model

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

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

print(f"Mean Squared Error: {mse:.2f}")

print(f"R^2 Score: {r2:.2f}")

# Plot Actual vs Predicted

plt.figure(figsize=(8,6))

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

plt.xlabel("Actual Prices ($1000's)")

plt.ylabel("Predicted Prices ($1000's)")

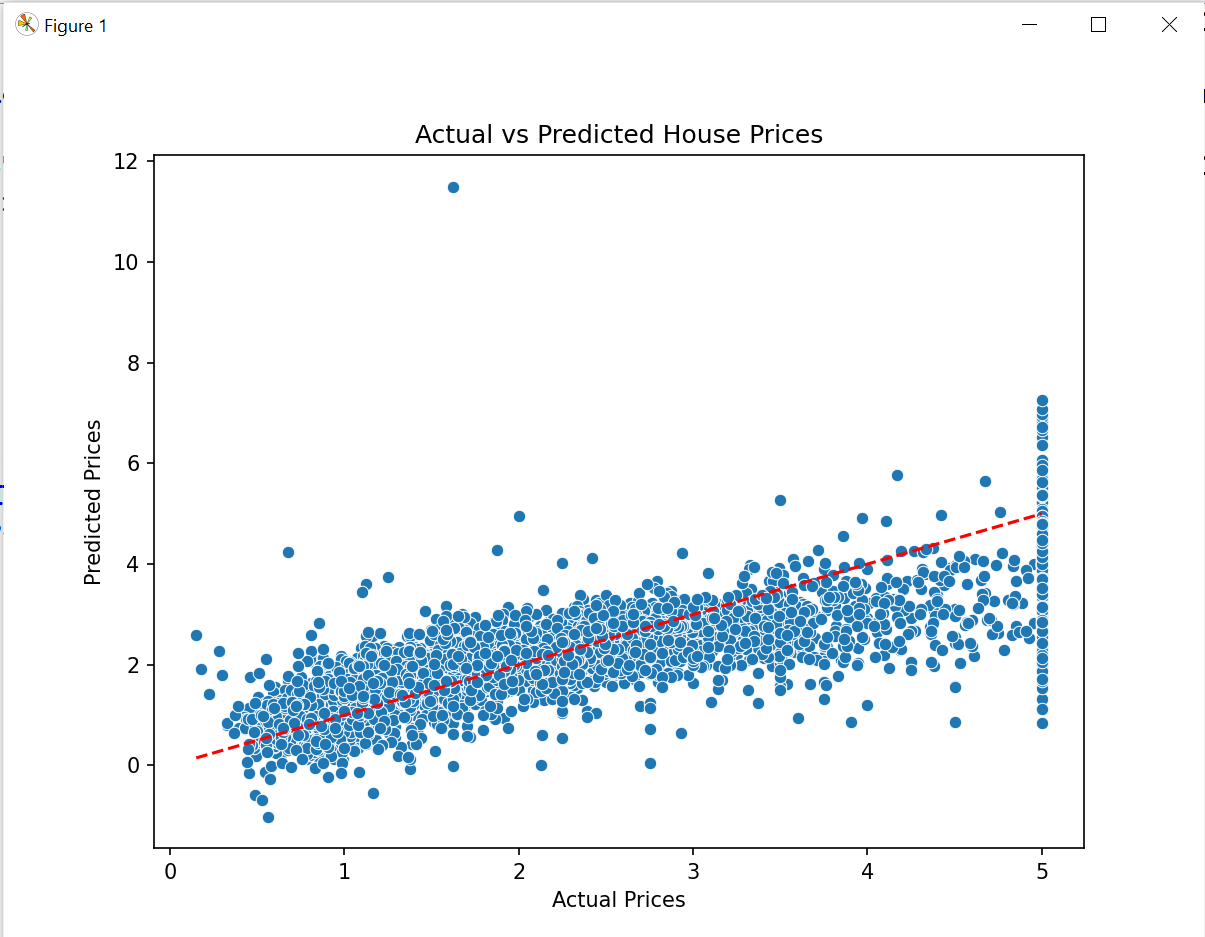
plt.title("Actual vs Predicted House Prices")

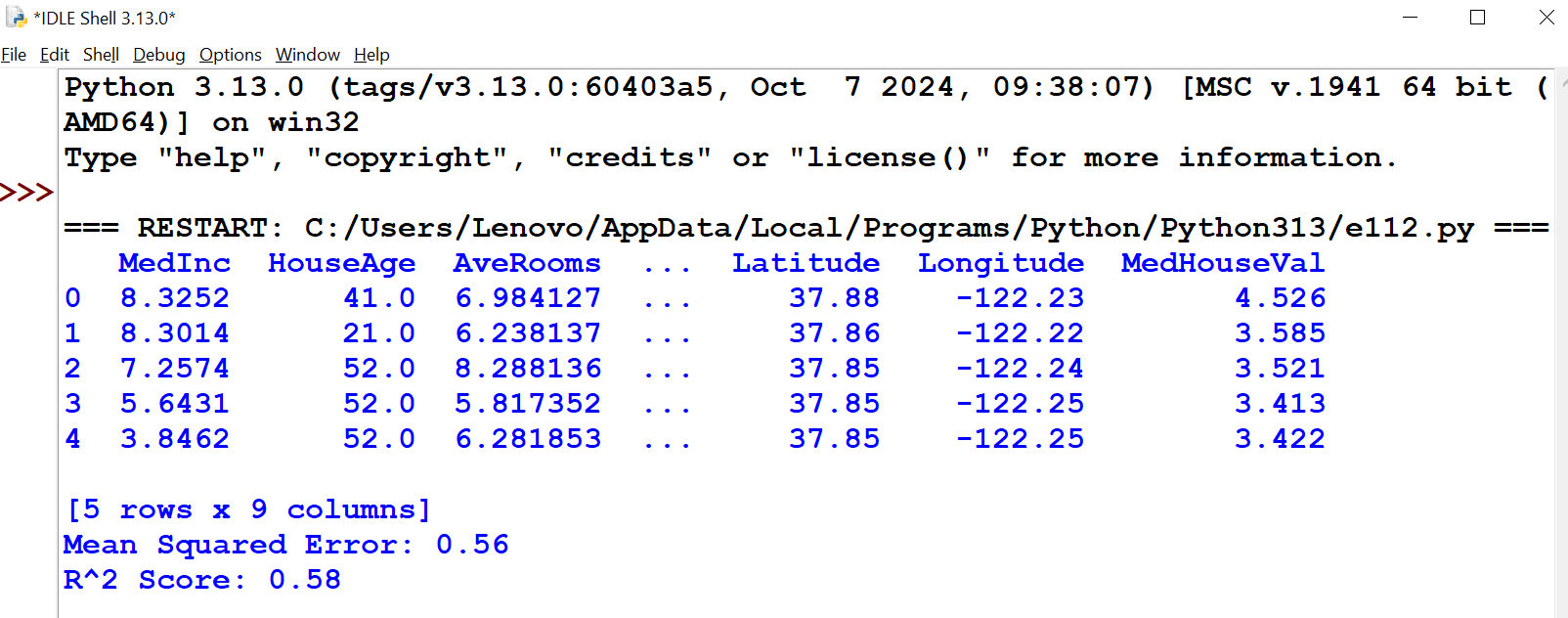
plt.plot([0, 50], [0, 50], '--r') # reference line

plt.show()

**📊 Explanation:**

* **Load Dataset:** Loads Boston housing data into a pandas DataFrame.
* **Train/Test Split:** 80% training, 20% test.
* **Linear Regression:** Simple linear regression model.
* **Evaluation:** Using MSE and R² score.
* **Visualization:** Scatter plot to compare actual vs predicted values





**CHANGED PROGRAM**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.linear\_model import LinearRegression

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

# Load the California Housing dataset

california = fetch\_california\_housing(as\_frame=True)

df = california.frame

# Display first 5 rows

print(df.head())

# Split data into features and target

X = df.drop('MedHouseVal', axis=1) # 'MedHouseVal' is the target variable

y = df['MedHouseVal']

# Split into training and test datasets

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

# Create Linear Regression model

model = LinearRegression()

# Train the model

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

# Predict on test set

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

# Evaluate the model

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

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

print(f"Mean Squared Error: {mse:.2f}")

print(f"R^2 Score: {r2:.2f}")

# Plot Actual vs Predicted

plt.figure(figsize=(8,6))

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

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual vs Predicted House Prices")

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], '--r') # Reference line

plt.show()