Linear Regression using Scikit-Learn

Project Description

This project demonstrates the implementation of **Linear Regression** for predicting house prices based on square footage. It includes **data loading, preprocessing, model training, evaluation, and visualization**.

Prerequisites

Required Libraries

- **Python 3.7 or later**
- `numpy` : For numerical computations.
- `pandas` : For data manipulation and analysis.
- `scikit-learn` : For machine learning algorithms and evaluation metrics.
- `matplotlib` : For data visualization.
- `seaborn` : For advanced visualizations.

Installation

Run the following command to install the necessary libraries:

```
pip install numpy pandas scikit-learn matplotlib seaborn
```

Code Description

Steps in the Code

1. Importing Libraries

```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

This section imports the required libraries.

---

2. Data Loading

```
 # Load dataset (Example: Housing Prices)
df = pd.read_csv("housing.csv")  # Replace with actual dataset

# Display first few rows
df.head()
```

The dataset is loaded using pandas. Replace `'housing.csv'` with the actual path to your dataset.

---

3. Minimal Processing

```
# Check for missing values
print(df.isnull().sum())

# Drop rows with missing values (if any)
df = df.dropna()

# Selecting feature(s) and target variable
X = df[['SquareFeet']]  # Independent variable
y = df['Price']  # Dependent variable

# Splitting data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

- **Handles missing values** by dropping them.
- **Splits the dataset** into 80% training and 20% testing.

---

4. Model Building

```
# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Display model coefficients
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

- Initializes a **Linear Regression model**.
- Trains the model on `X_train` and `y_train`.
- Displays **intercept** and **coefficient(s)**.

---

5. Making Predictions

```
# Predict on test data
y_pred = model.predict(X_test)

# Display actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

- The model predicts **house prices** for the test data.
- Displays actual vs predicted values.

---

6. Performance Metrics

```
# Calculate and print performance metrics
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"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
print(f"R-squared (R²): {r2}")
```

- **Mean Absolute Error (MAE)**: Measures the average absolute errors.
- **Mean Squared Error (MSE)**: Measures squared differences.
- **Root Mean Squared Error (RMSE)**: Measures standard deviation of prediction errors.
- **R-squared (R^2)**: Measures how well the model explains variance.

---

7. Visualization

Scatter Plot: Actual vs Predicted Prices

```
# Scatter plot of actual vs predicted values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, color='blue', alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle=
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.show()
```

- **Scatter plot** comparing actual vs predicted prices.
- A **red dashed line** represents the ideal prediction.

Regression Line Visualization

```
# Visualizing the regression line
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_train.values.flatten(), y=y_train, color='blue', label="Training Da
sns.scatterplot(x=X_test.values.flatten(), y=y_test, color='green', label="Testing Data'
plt.plot(X_test.values.flatten(), y_pred, color='red', linewidth=2, label="Regression Li
plt.xlabel("Square Feet")
plt.ylabel("Price")
plt.title("Linear Regression Fit")
plt.legend()
plt.show()
```

- **Blue dots**: Training data.
- **Green dots**: Testing data.
- **Red line**: Regression line.

Outputs

Metrics:

- **Mean Absolute Error (MAE)**
- **Mean Squared Error (MSE)**
- **Root Mean Squared Error (RMSE)**
- **R-squared (R^2) Score**

Visualization:

- **Scatter plot of actual vs predicted prices**
- **Regression line over training & test data**

Example Output

Sample Model Coefficients:

```
 Intercept: 50000
Coefficient: [200]
```

Sample Predictions:

```
   Actual  Predicted
0  250000     248000
1  300000     302000
2  400000     395000
```

Sample Metrics:

```
 Mean Absolute Error (MAE): 4500.0
Mean Squared Error (MSE): 34000000.0
Root Mean Squared Error (RMSE): 5830.95
R-squared (R²): 0.87
```

Use Cases

This project is useful for:

- **Predicting housing prices** based on square footage.
- **Understanding regression analysis** in machine learning.
- **Evaluating model performance** using regression metrics.

Future Enhancements

- **Feature engineering**: Incorporate more variables like number of rooms, location, etc.
- **Hyperparameter tuning**: Optimize model for better accuracy.
- **Deploying the model**: Use Flask or FastAPI for web-based predictions.
