23CSE301 Machine Learning

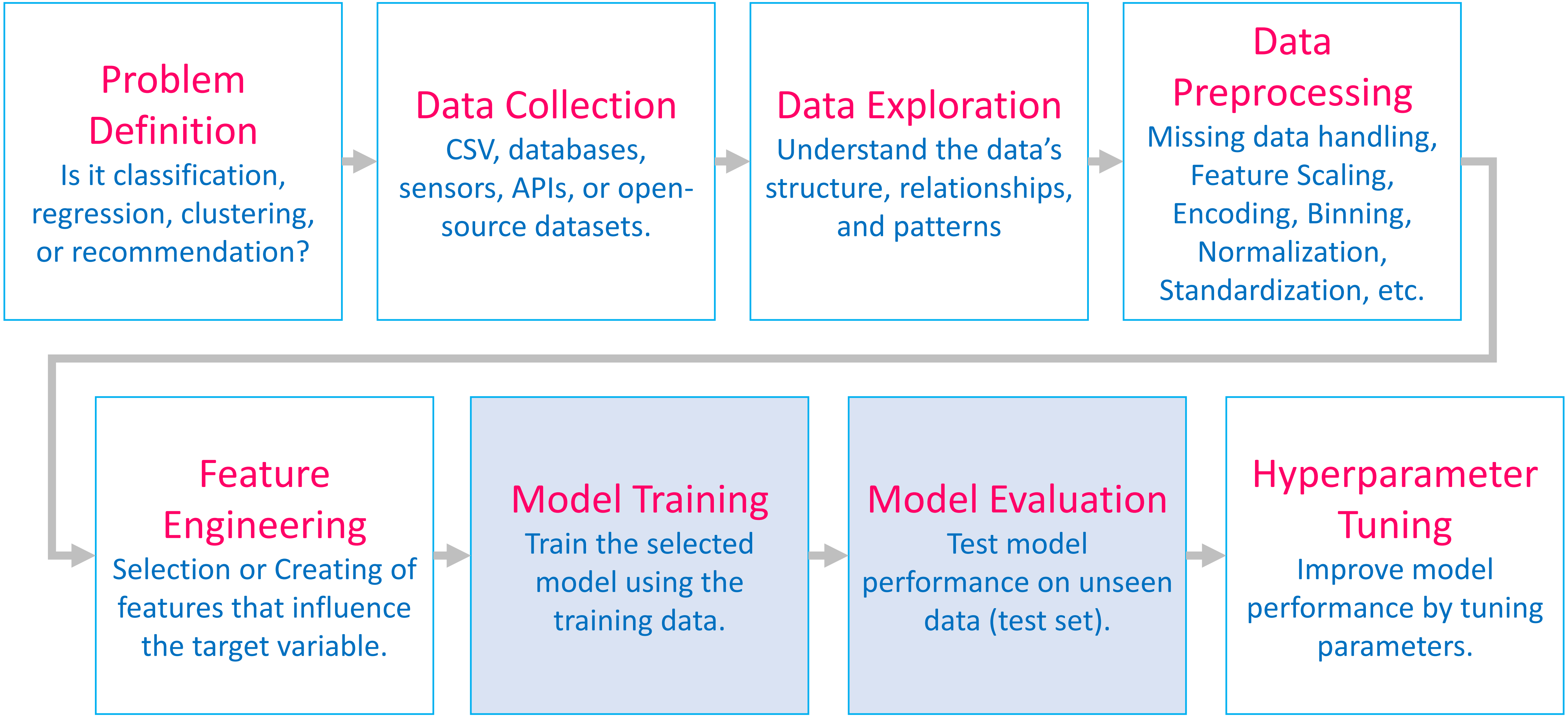
V Sem. CSE B

Practical – Week 6

Course Instructor: Dr. M. Anbazhagan

Pract. #	Experiment Title
P1-P3	• Introduction: Python, Pandas (scikit learn and other libraries) • Pre-processing: Dataset selection, Exploratory Data analysis and Feature engineering; Introduction to Colab/Jupyter Notebook, Pandas(Data Frames); Data Selection (iloc, loc); Sorting, Grouping merge, join, concat; Crosstab; Missing data treatment(fillna, dropna), Converting categorical values, Visualization(Line chart, Bar Chart, Pie chart, Scatter plot, Box plot); Distributions; Summary statistics.
Lab 1 Evaluation (P1 to P3)	
P4	Dimensionality Reduction Technique: PCA
P5	Feature Selection
P6	Regression Algorithms: Linear Regression
P7	Regression Algorithms: Logistic Regression
P8	Classification Algorithms: Decision Tree Classifier
	Classification Algorithms: K-Nearest Neighbor Classifier
Lab 2 Mid-Term exam (P1 to P8)	
P9	Classification Algorithms: Random Forest Classifier, ensemble learning.
P10	Classification Algorithms: Support Vector Machines
P11	Classification Algorithms: Perceptron
P12	Clustering: 1. K-Means Clustering 2. Agglomerative Clustering
Lab 3 Evaluation (P1 to P12)	

End-to-End Machine Learning Pipeline



Linear Regression

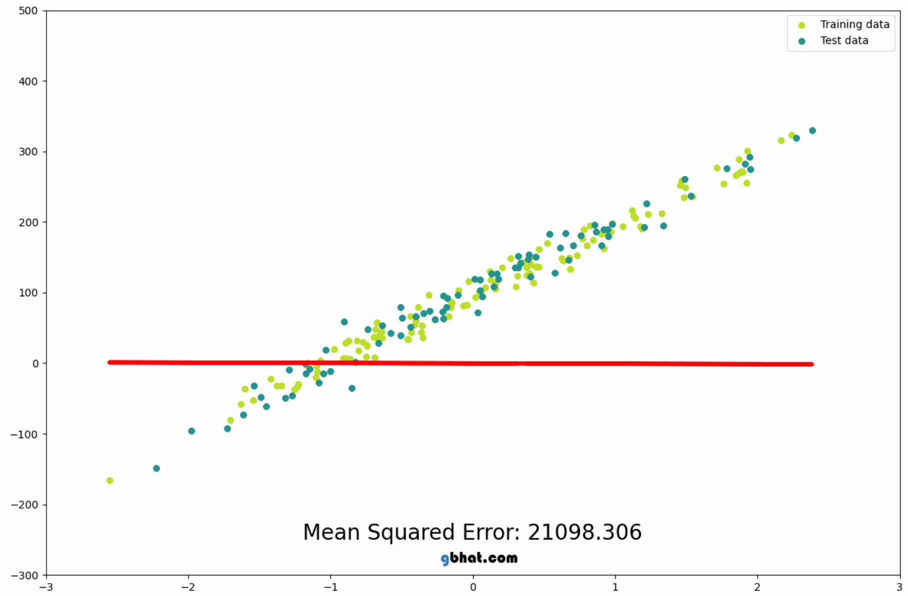
The simplest and most widely used supervised learning algorithms in machine learning, used for predicting a continuous output variable based on one or more input features.

$$y = \beta_0 + \beta_1 x + \epsilon$$

Simple Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Multiple Linear Regression



Linear Regression in Scikit-learn

Linear Regression is implemented via the sklearn.linear_model.LinearRegression class

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

Attribute	Description
coef_	Coefficients for each feature
intercept_	The bias term
score()	R^2 score
predict()	Predicts target values for new inputs

Modeling a Linear Regressor

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

Used to import a Linear Regression model from the scikit-learn library.

Used to split your dataset into training and testing sets.

Used to evaluate how well your model performs.

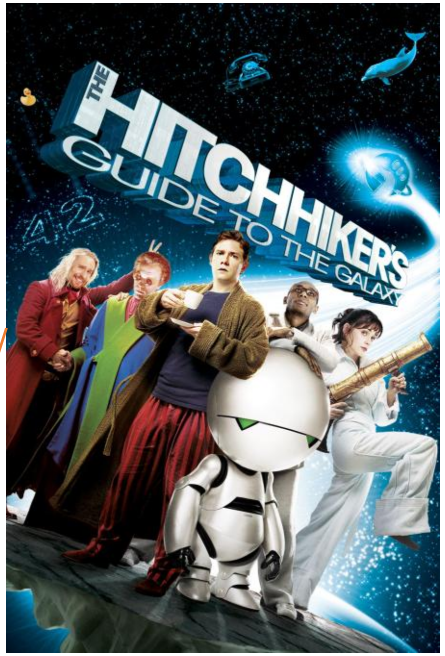
```
# Load sample dataset
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
```

Used to load a real-world dataset of California housing prices.

- data.data
- data.target
- data.frame

Modeling a Linear Regressor

```
# Preview
print(df.head())

# Select one feature (e.g., MedInc) for Simple Linear Regression
X = df[['MedInc']]   # median income
y = df['MedHouseVal']  # median house value
```




```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- X_train, y_train: 80% of your data used for model training
- X_test, y_test: 20% held back to evaluate the model’s performance

Modeling a Linear Regressor

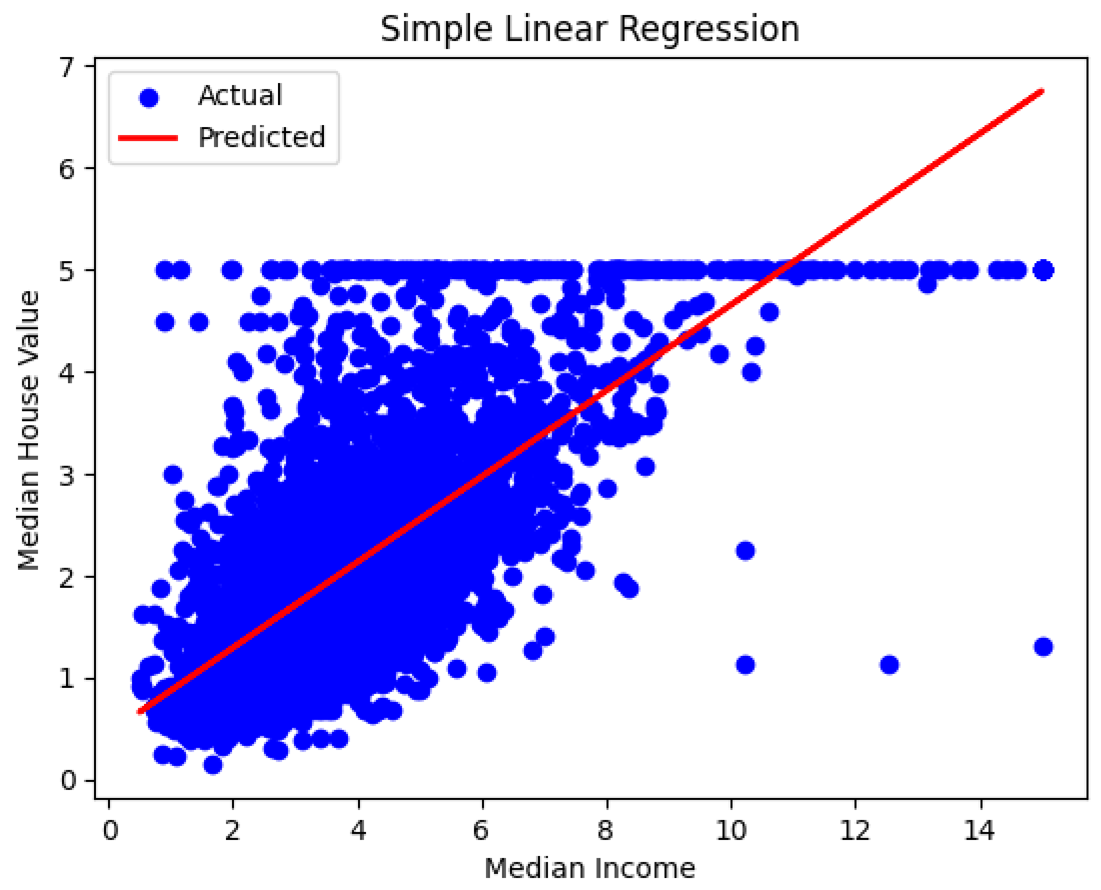
```
# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
```

```
# Evaluation
print("Coefficient (slope):", model.coef_)
print("Intercept:", model.intercept_)
print("R² score:", r2_score(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
```

```
Coefficient (slope): 0.4205545738380291
Intercept: 0.4472183362106792
R² score: 0.4719083593446772З
MSE: 0.692692969609108
```

Modeling a Linear Regressor

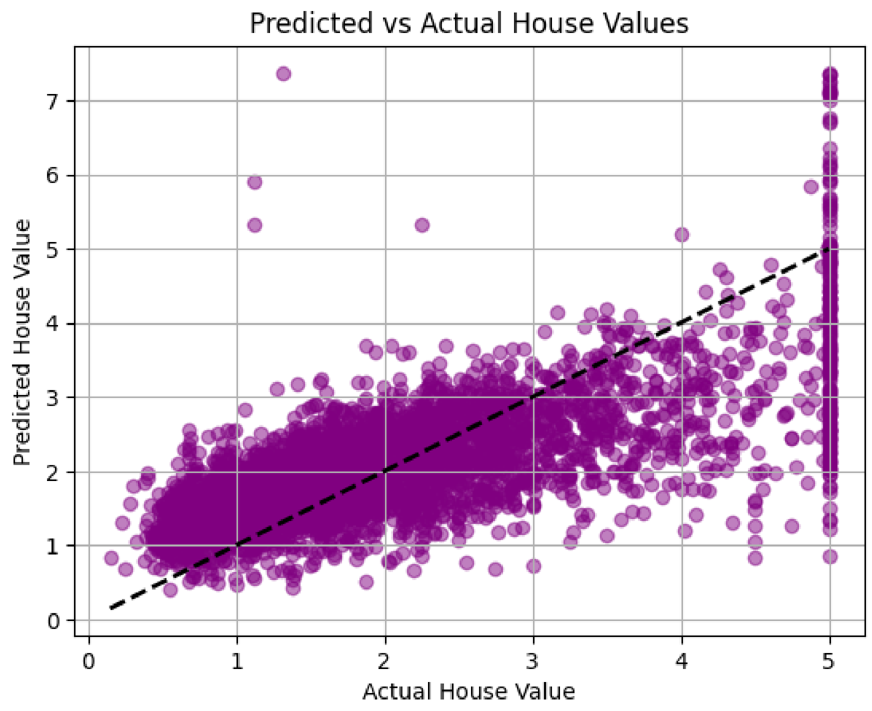
```
# Plot
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.legend()
plt.title("Simple Linear Regression")
plt.show()
```



Modeling a Linear Regressor

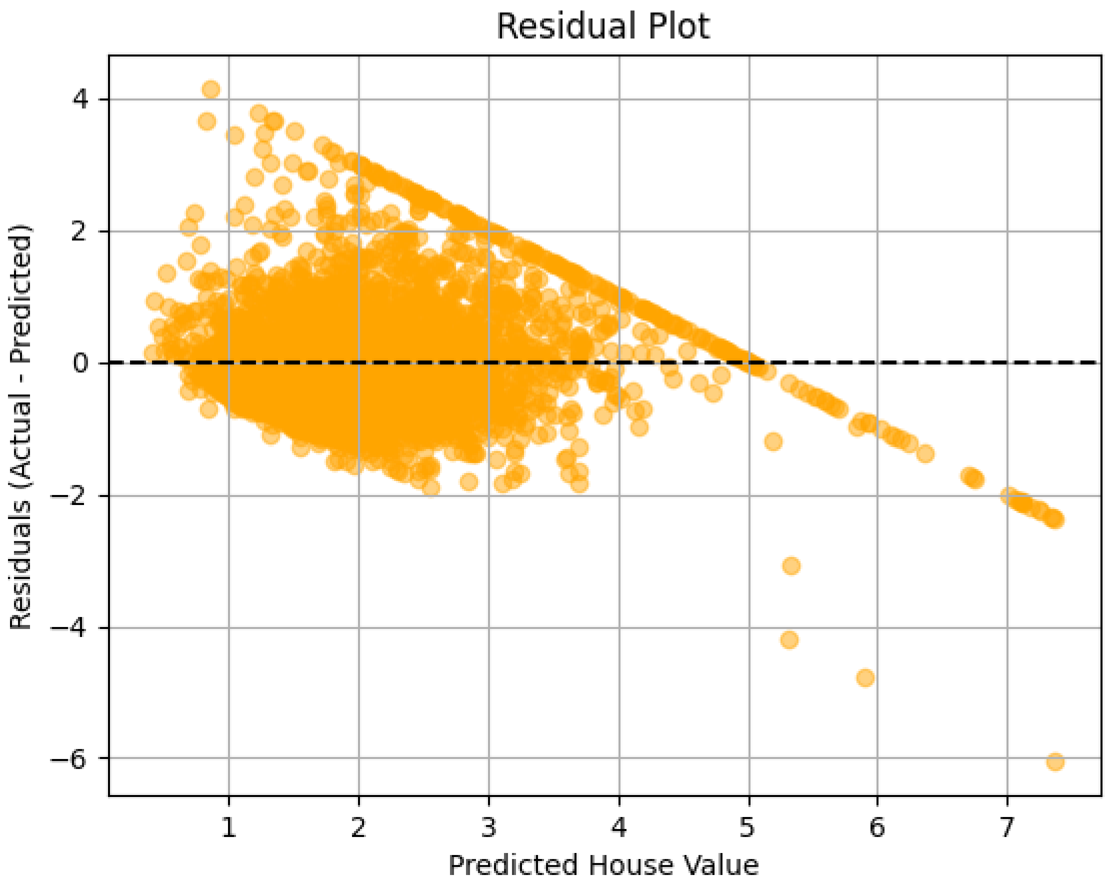
```
# Select two or three features for Multiple Linear Regression
X = df[['MedInc', 'AveRooms', 'HouseAge']]
y = df['MedHouseVal']
```

```
# Predicted vs. Actual Scatter Plot
plt.scatter(y_test, y_pred, color='purple', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual House Value')
plt.ylabel('Predicted House Value')
plt.title('Predicted vs Actual House Values')
plt.grid(True)
plt.show()
```



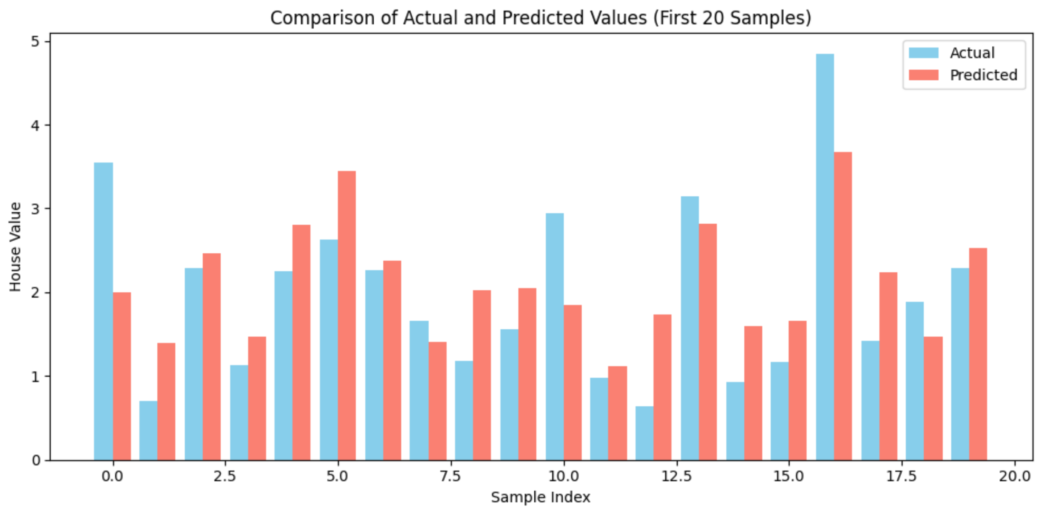
Modeling a Linear Regressor

```
# Residual Plot
residuals = y_test - y_pred


plt.scatter(y_pred, residuals, color='orange', alpha=0.5)
plt.axhline(y=0, color='black', linestyle='--')
plt.xlabel('Predicted House Value')
plt.ylabel('Residuals (Actual - Predicted)')
plt.title('Residual Plot')
plt.grid(True)
plt.show()
```



Modeling a Linear Regressor

```
# Bar Chart of Sample Predictions
n = 20  # show 20 predictions
indices = np.arange(n)
plt.figure(figsize=(10, 5))
plt.bar(indices - 0.2, y_test[:n], width=0.4, label='Actual', color='skyblue')
plt.bar(indices + 0.2, y_pred[:n], width=0.4, label='Predicted', color='salmon')
plt.xlabel('Sample Index')
plt.ylabel('House Value')
plt.title('Comparison of Actual and Predicted Values (First 20 Samples)')
plt.legend()
plt.tight_layout()
plt.show()
```



Gradient Descent Optimizer for LinReg

By default, scikit-learn's Linear Regression does NOT use Gradient Descent Optimizer (it uses OLS).

```
from sklearn.linear_model import SGDRegressor
model = SGDRegressor()
model.fit(X_train, y_train)
```

Attribute	Description
coef_	Coefficients for each feature
intercept_	The bias term
score()	R^2 score
predict()	Predicts target values for new inputs

Modeling a SGDRegressor

```
# Changed and Additional Dependencies
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
```

```
# Scaling (IMPORTANT for gradient descent!)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# SGD Regressor (Gradient Descent-based Linear Regression)
model = SGDRegressor(max_iter=100, learning_rate='invscaling', eta0=0.01, random_state=42)
model.fit(X_train_scaled, y_train)
```

Maximum number of passes over the training data. Stops earlier if convergence is reached.

The learning rate decreases as training progresses

Initial learning rate (η_0). It starts at 0.01 and gets scaled down depending on the learning_rate schedule.

Exercise 1 - Week 6

- Dataset: Advertising dataset
  - Using the Advertising dataset, perform a comprehensive linear regression analysis. Begin with data preprocessing by checking for missing values, basic statistics, and visualizing distributions. Split the data into training and test sets using three different ratios (80:20, 70:30, and 60:40) and record the split sizes. First, build a simple linear regression model using TV advertising spend to predict Sales. Then, extend to multiple linear regression using all three features: TV, Radio, and Newspaper. For both models, train, predict, and evaluate using metrics like R^2, MSE, and MAE. Finally, visualize the actual vs. predicted values and residuals, and conclude with a brief analysis comparing the two models and the effect of varying test sizes.
  - Note: Inference can be written into a text/comment cell at the very last.

Exercise 2 - Week 6

- Dataset: Student Performance dataset
  - Using the Student Performance dataset, analyze and predict the final grade (G3) using linear regression techniques. Start by preprocessing the data: handle missing values, encode categorical variables, and explore correlations. First, implement simple linear regression using a single relevant feature (e.g., study time or G1). Then perform multiple linear regression using a subset of significant predictors (e.g., G1, G2, study time, failures). In addition, train a linear regression model using Stochastic Gradient Descent (SGDRegressor) and compare its performance with OLS-based models. Evaluate all models using R^2, MSE, and MAE, and visualize actual vs. predicted grades and residuals. Conclude with a discussion on model comparisons, the effect of different splits, and the advantages of using SGD.
  - Note: Inference can be written into a text/comment cell at the very last.