# Day45_Stop_Overfitting_With_L1_Lasso_and_L2_Ridge

July 15, 2025

**Day 45 – Regularization Techniques in Machine Learning (L1, L2, ElasticNet)**

This notebook demonstrates how **L1 and L2 regularization** techniques help **prevent overfitting** in Machine Learning models.

We'll explore:

- What overfitting is and why it's bad
- How **Ridge (L2)** shrinks coefficients (keeps all features)
- How **Lasso (L1)** removes unnecessary features (sets some coefficients to 0)
- Visual and numeric comparison of Linear, Ridge, and Lasso regression
- Model evaluation using **$R^2$**, **Adjusted $R^2$**, **RMSE**, and **Residual Plots**

**Outcome:**

By the end, you'll understand:

- How to choose between Ridge and Lasso
- When regularization improves your model
- Why simpler models often perform better on unseen data

> **Use L1 & L2 to make your models more robust, generalizable, and interpretable!**

**What is Machine Learning? (Recap)**

Machine Learning (ML) is a branch of Artificial Intelligence where computers learn from data to make predictions or decisions without being explicitly programmed.

Machine Learning is like teaching a computer how to learn from past examples — just like humans learn from experience. For example:

- If a kid sees that touching fire hurts, they learn not to do it again.

- ML algorithms do the same — they find patterns from data to predict or decide things.

**Types of Machine Learning:**

- Supervised Learning (Labeled data):
  - Like a teacher guiding. You give input + correct answer.
    - e.g., Regression (Predicting prices), Classification (classifying emails.)
- Unsupervised Learning (Unlabeled data):
  - No answers, the algorithm finds patterns on its own.
    - e.g., Clustering (Grouping similar customers.)
- Reinforcement Learning:

  - Learning by reward and punishment.
    * e.g., Games, robots learning to walk.

**Linear Regression - Predicting with Lines**

Regression is about predicting numbers. For example, predicting:

- Salary based on years of experience
- Mileage (mpg) of a car based on features like horsepower, weight, etc.
- Simple Linear Regression:
  - Predicts using one feature → Equation:

$$y = mx + b$$

- Multiple Linear Regression (MLR):
  - Uses multiple features →

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

**Underfitting vs Overfitting**

- Underfitting: Model is too simple → Poor on both train/test (misses patterns)
- Overfitting: Model is too complex → Great on training, bad on test (memorizes data)

**Techniques to Handle Overfitting**

- Regularization
- Cross-validation
- Simpler models
- More data
- Dropout
- Pruning

**What is Regularization?**

Regularization helps prevent overfitting by penalizing large coefficients in the model.

**Types of Regularization**

| Type | Penalty | Behavior |
|---|---|---|
| **Ridge** | L2 | Shrinks coefficients (not zero) |
| **Lasso** | L1 | Some coefficients become 0 |
| **ElasticNet** | L1 + L2 | Combines both |

**Analogy:**

- Lasso = Removing unhelpful friends
- Ridge = Asking loud people to speak softer

**Steps in This Notebook:**

- Build baseline MLR model
- Apply Lasso, Ridge, and ElasticNet
- Compare performance

## 1 Import Required Libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score
```

## 2 Load and Explore the Dataset

[2]:
```
data = pd.read_csv(r"C:\Users\Lenovo\Downloads\car-mpg.csv")
data.head()
```

[2]:
```
    mpg  cyl   disp   hp    wt   acc  yr  origin  car_type  \
0  18.0    8  307.0  130  3504  12.0  70       1         0
1  15.0    8  350.0  165  3693  11.5  70       1         0
2  18.0    8  318.0  150  3436  11.0  70       1         0
3  16.0    8  304.0  150  3433  12.0  70       1         0
4  17.0    8  302.0  140  3449  10.5  70       1         0

                    car_name
0  chevrolet chevelle malibu
1          buick skylark 320
2         plymouth satellite
3              amc rebel sst
4                ford torino
```

## 3 Data Cleaning & Encoding

- Drop irrelevant columns (`car_name`)
- Convert and encode `origin`
- Replace `?` with `NaN`, fill with **median**

### 3.1 Drop car_name as it's not a useful feature

[3]:
```python
# Drop car_name as it's not a useful feature
data = data.drop(['car_name'], axis=1)
```

### 3.2 Convert 'origin' to categorical labels

[4]:
```python
# Convert 'origin' to categorical labels
data['origin'] = data['origin'].replace({1: 'america', 2: 'europe', 3: 'asia'})
```

### 3.3 One-hot encode the 'origin' column

[5]:
```python
# One-hot encode the 'origin' column
data = pd.get_dummies(data, columns=['origin'],dtype=int)
```

### 3.4 Replace '?' with NaN and then fill missing values with median

[6]:
```python
# Step 1: Replace '?' with NaN (if applicable)
data = data.replace('?', np.nan)

# Step 2: Convert all columns to numeric where possible
data = data.apply(pd.to_numeric, errors='coerce')

# Step 3: Select only numeric columns
numeric_cols = data.select_dtypes(include=[np.number]).columns

# Step 4: Fill missing values in numeric columns with the median
data[numeric_cols] = data[numeric_cols].apply(lambda x: x.fillna(x.median()))
```

[7]:
```python
data.head()
```

[7]:
```
    mpg  cyl   disp     hp    wt   acc  yr  car_type  origin_america  \
0  18.0    8  307.0  130.0  3504  12.0  70         0               1
1  15.0    8  350.0  165.0  3693  11.5  70         0               1
2  18.0    8  318.0  150.0  3436  11.0  70         0               1
3  16.0    8  304.0  150.0  3433  12.0  70         0               1
4  17.0    8  302.0  140.0  3449  10.5  70         0               1

   origin_asia  origin_europe
0            0              0
1            0              0
2            0              0
3            0              0
4            0              0
```

## 4 Feature and Target Separation

[8]:
```
X = data.drop(['mpg'], axis=1) # Independent Variables
y = data[['mpg']] # Dependent Variable
```

## 5 Scaling

Scale columns so none dominate (e.g., weight vs hp)

[9]:
```
# Scale the data
X_s = preprocessing.scale(X)
X_s = pd.DataFrame(X_s, columns=X.columns)
```

[10]:
```
X_s
```

[10]:
```
          cyl      disp        hp        wt       acc        yr  car_type  \
0    1.498191  1.090604  0.673118  0.630870 -1.295498 -1.627426 -1.062235
1    1.498191  1.503514  1.589958  0.854333 -1.477038 -1.627426 -1.062235
2    1.498191  1.196232  1.197027  0.550470 -1.658577 -1.627426 -1.062235
3    1.498191  1.061796  1.197027  0.546923 -1.295498 -1.627426 -1.062235
4    1.498191  1.042591  0.935072  0.565841 -1.840117 -1.627426 -1.062235
..        ...       ...       ...       ...       ...       ...       ...
393 -0.856321 -0.513026 -0.479482 -0.213324  0.011586  1.621983  0.941412
394 -0.856321 -0.925936 -1.370127 -0.993671  3.279296  1.621983  0.941412
395 -0.856321 -0.561039 -0.531873 -0.798585 -1.440730  1.621983  0.941412
396 -0.856321 -0.705077 -0.662850 -0.408411  1.100822  1.621983  0.941412
397 -0.856321 -0.714680 -0.584264 -0.296088  1.391285  1.621983  0.941412

     origin_america  origin_asia  origin_europe
0          0.773559    -0.497643      -0.461968
1          0.773559    -0.497643      -0.461968
2          0.773559    -0.497643      -0.461968
3          0.773559    -0.497643      -0.461968
4          0.773559    -0.497643      -0.461968
..              ...          ...            ...
393        0.773559    -0.497643      -0.461968
394       -1.292726    -0.497643       2.164651
395        0.773559    -0.497643      -0.461968
396        0.773559    -0.497643      -0.461968
397        0.773559    -0.497643      -0.461968

[398 rows x 10 columns]
```

[11]:
```
y_s = preprocessing.scale(y)
y_s = pd.DataFrame(y_s, columns=y.columns)
```

[12]:
```
y_s
```

[12]:
```
          mpg
0   -0.706439
1   -1.090751
2   -0.706439
3   -0.962647
4   -0.834543
..        …
393  0.446497
394  2.624265
395  1.087017
396  0.574601
397  0.958913

[398 rows x 1 columns]
```

# 6 Train test split

70% training, 30% testing for validation

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X_s, y_s, test_size=0.3,␣
 ↪random_state=1)
```

# 7 Train a Simple Linear Regression Model

**Uses all features directly**

[14]:
```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)

# Print coefficients and intercept
for idx, col_name in enumerate(X_train.columns):
    print(f"The coefficient for {col_name} is {regression_model.coef_[0][idx]}")

print("Intercept:", regression_model.intercept_[0])
```

```
The coefficient for cyl is 0.321022385691611
The coefficient for disp is 0.32483430918483897
The coefficient for hp is -0.22916950059437569
The coefficient for wt is -0.7112101905072298
The coefficient for acc is 0.014713682764191237
The coefficient for yr is 0.37558119495107448
The coefficient for car_type is 0.3814769484233099
The coefficient for origin_america is -0.07472247547584178
The coefficient for origin_asia is 0.044515252035677896
The coefficient for origin_europe is 0.04834854953945386
Intercept: 0.019284116103639764
```

# 8 Ridge Regression (L2 Regularization)

**Shrinks coefficients to control overfitting**

[15]:
```
ridge_model = Ridge(alpha=0.3)
ridge_model.fit(X_train, y_train)

print("Ridge model coefficients:", ridge_model.coef_)
```

```
Ridge model coefficients: [ 0.31649043  0.31320707 -0.22876025 -0.70109447
0.01295851  0.37447352
  0.37725608 -0.07423624  0.04441039  0.04784031]
```

# 9 Lasso Regression (L1 Regularization)

**Drops unnecessary features by zeroing out coefficients**

[16]:
```
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)

print("Lasso model coefficients:", lasso_model.coef_)
```

```
Lasso model coefficients: [-0.          -0.          -0.01690287 -0.51890013  0.
0.28138241
  0.1278489  -0.01642647  0.          0.        ]
```

# 10 Tips

- Use $R^2$ when starting
- Use Adjusted $R^2$ when comparing models with different number of features
- Use RMSE (or MAE) to see real prediction error
- Use visuals (like scatter plots & residual plots) to validate behavior

# 11 $R^2$ Score Comparison

**$R^2$ (Coefficient of Determination)** tells us how much variation in the target (e.g. `mpg`) is explained by the model.

- $R^2 = 1$: Perfect prediction
- $R^2 = 0$: Model explains nothing
- $R^2 < 0$: Worse than just predicting the mean
- **Ideal range**: Closer to **1** is better

[17]:
```python
print("Linear Train R²:", regression_model.score(X_train, y_train))
print("Linear Test R²:", regression_model.score(X_test, y_test))

print("Ridge Train R²:", ridge_model.score(X_train, y_train))
print("Ridge Test R²:", ridge_model.score(X_test, y_test))

print("Lasso Train R²:", lasso_model.score(X_train, y_train))
print("Lasso Test R²:", lasso_model.score(X_test, y_test))
```

```
Linear Train R²: 0.8343770256960538
Linear Test R²: 0.8513421387780066
Ridge Train R²: 0.8343617931312616
Ridge Test R²: 0.8518882171608506
Lasso Train R²: 0.7938010766228453
Lasso Test R²: 0.8375229615977083
```

### 11.1 Interpretation:

- All models do well ($R^2 > 0.79$).
- Ridge gives the best generalization.
- Lasso is slightly less accurate, but it simplifies the model by eliminating less useful features.

## 12 Adjusted $R^2$ Using statsmodels (Like in R)

Adjusted $R^2$ improves over plain $R^2$ by considering how many predictors you're using. It penalizes unnecessary features.

- Adjusted $R^2 = 1 \rightarrow$ Perfect prediction (just like $R^2$)
- Adjusted $R^2 < R^2 \rightarrow$ Penalizes too many features
- Adjusted $R^2 < 0 \rightarrow$ Model likely overfits or uses irrelevant features
- Best for comparing models with **different numbers of predictors**
- **Closer to 1 is better** — just like $R^2$, but more honest!

[18]:
```python
import statsmodels.formula.api as smf

# Merge features and target
data_train_test = pd.concat([X_train, y_train], axis=1)

# Fit OLS regression model
ols1 = smf.ols(formula='mpg ~␣
↪cyl+disp+hp+wt+acc+yr+car_type+origin_america+origin_europe+origin_asia',␣
↪data=data_train_test).fit()
```

```python
# View summary
print(ols1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.834
Model:                            OLS   Adj. R-squared:                  0.829
Method:                 Least Squares   F-statistic:                     150.0
Date:                Tue, 15 Jul 2025   Prob (F-statistic):           3.12e-99
Time:                        13:30:17   Log-Likelihood:                -146.89
No. Observations:                 278   AIC:                             313.8
Df Residuals:                     268   BIC:                             350.1
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
Intercept          0.0193      0.025      0.765      0.445      -0.030
0.069
cyl                0.3210      0.112      2.856      0.005       0.100
0.542
disp               0.3248      0.128      2.544      0.012       0.073
0.576
hp                -0.2292      0.079     -2.915      0.004      -0.384
-0.074
wt                -0.7112      0.088     -8.118      0.000      -0.884
-0.539
acc                0.0147      0.039      0.373      0.709      -0.063
0.092
yr                 0.3756      0.029     13.088      0.000       0.319
0.432
car_type           0.3815      0.067      5.728      0.000       0.250
0.513
origin_america    -0.0747      0.020     -3.723      0.000      -0.114
-0.035
origin_europe      0.0483      0.021      2.270      0.024       0.006
0.090
origin_asia        0.0445      0.020      2.175      0.031       0.004
0.085
==============================================================================
Omnibus:                       22.678   Durbin-Watson:                   2.105
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               36.139
Skew:                           0.513   Prob(JB):                     1.42e-08
Kurtosis:                       4.438   Cond. No.                     1.59e+16
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 6.14e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

### 12.1 Adjusted $R^2$ from statsmodels:

```
R-squared:      0.834
Adj. R-squared: 0.829
```

- Close values → most features are useful
- If Adjusted $R^2$ was much lower than $R^2$, it would mean too many unhelpful features

## 13 Root Mean Squared Error (RMSE)

RMSE shows the average prediction error in the same units as the target variable.

- RMSE $\geq 0 \rightarrow$ Cannot be negative
- RMSE $= 0 \rightarrow$ Perfect prediction (no error)
- The **lower**, the better
- Same unit as the predicted variable (e.g., mpg, price, etc.)

  Example: RMSE $= 2.5 \rightarrow$ on average, your predictions are off by 2.5 units

[19]:
```
mse = np.mean((regression_model.predict(X_test) - y_test) ** 2)
import math
rmse = math.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 0.37766934254087847
```

Very low RMSE! Your model is off by just ±0.38 on average when predicting mpg.

## 14 Residual Plot – Check for Patterns

**Why plot it?**

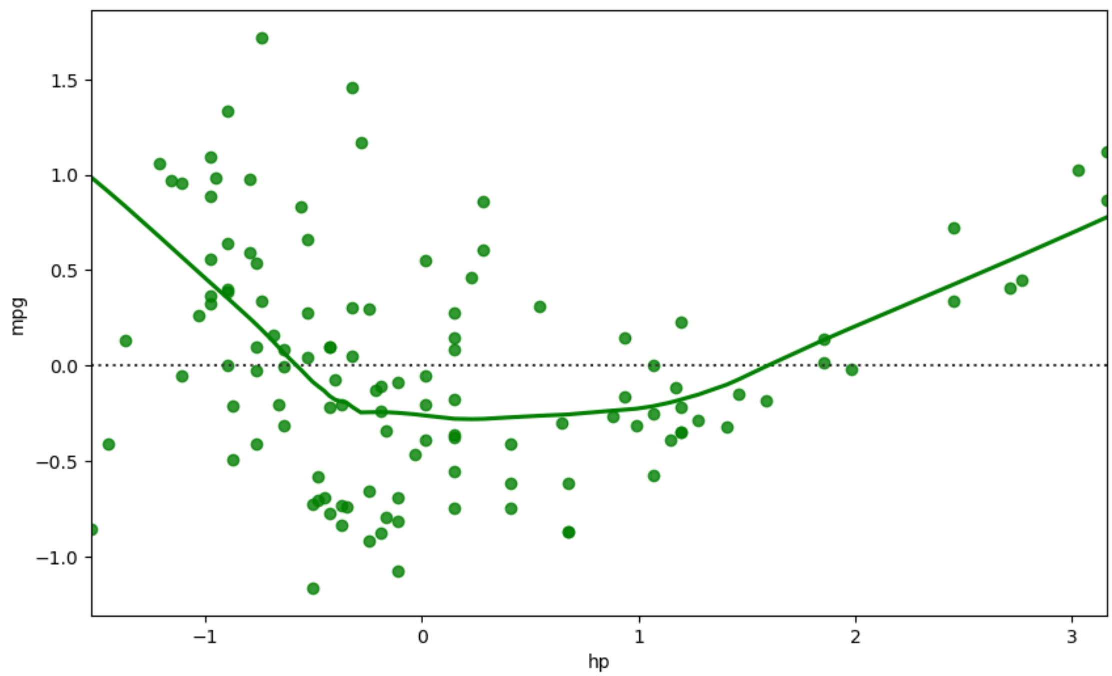
- To check if the model is missing any patterns
- A good residual plot has no shape — it looks random

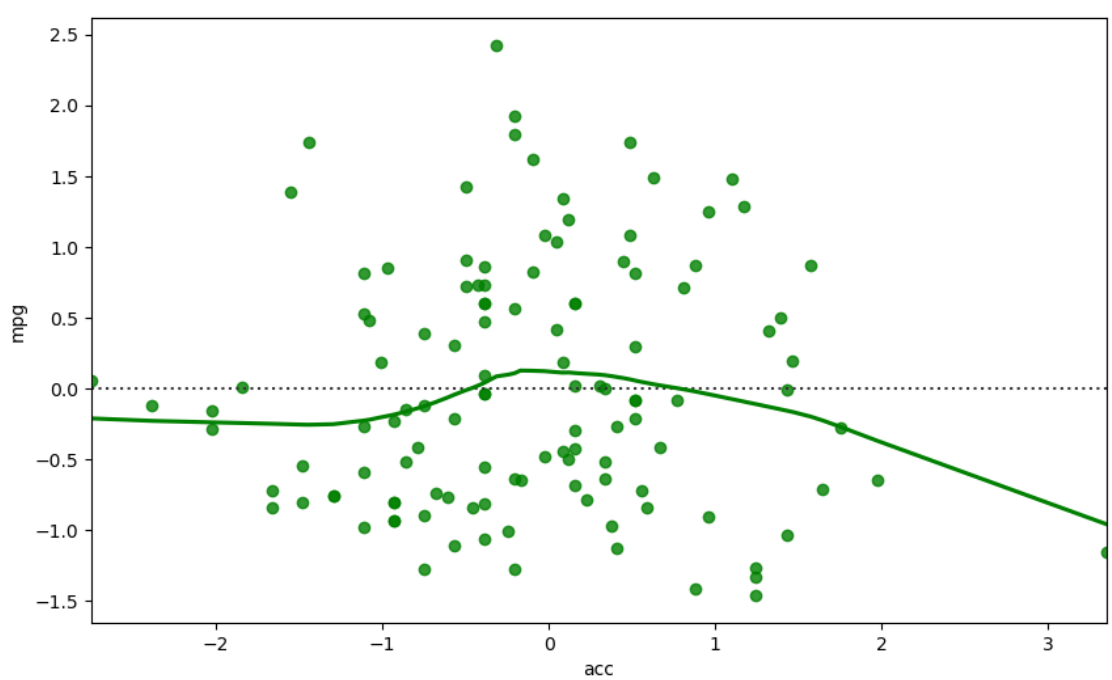
**Our plots look random → model fits well**

[20]:
```
fig = plt.figure(figsize=(10, 6))
sns.residplot(x=X_test['hp'], y=y_test['mpg'], color='green', lowess=True)
```

```
fig = plt.figure(figsize=(10, 6))
sns.residplot(x=X_test['acc'], y=y_test['mpg'], color='green', lowess=True)
```

[20]: <Axes: xlabel='acc', ylabel='mpg'>

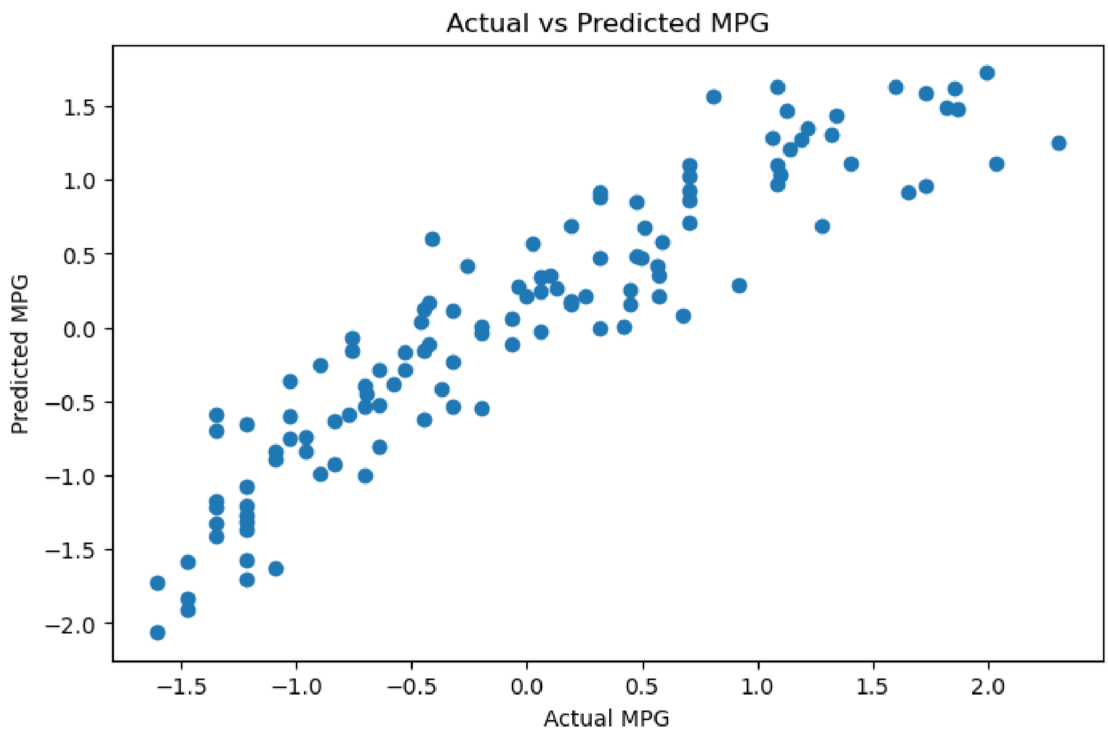


## 15 Prediction vs Actual – Visualization

[21]:
```
y_pred = regression_model.predict(X_test)

plt.figure(figsize=(8, 5))
plt.scatter(y_test['mpg'], y_pred)
plt.xlabel("Actual MPG")
plt.ylabel("Predicted MPG")
plt.title("Actual vs Predicted MPG")
plt.show()
```




**This plot compares:**

Actual mpg values vs predicted mpg

- A good model will show points clustered around the diagonal line
- Your plot shows this — predictions are close to real-world values

## 16 Final Summary

- **Linear Regression** is simple and accurate, but it may **overfit** when the model becomes too complex or when irrelevant features are included.

- **Ridge Regression (L2)** is the **best performer** in this case — it shrinks coefficients slightly to prevent overfitting while maintaining good accuracy and generalization.
- **Lasso Regression (L1)** is slightly less accurate but more **interpretable** — it can automatically **remove weak or unnecessary features** by setting their coefficients to zero.
- **Adjusted $R^2$** helps evaluate model quality more fairly when you have many features — it penalizes complexity.
- **RMSE (Root Mean Squared Error)** tells how far your predictions are from actual values — lower RMSE is better.
- **Residual and Scatter Plots** help visualize errors and check how well your model fits unseen data.

### 16.1 Recommendation:

Use a combination of **$R^2$**, **Adjusted $R^2$**, **RMSE**, and **visual plots** to compare and select the most reliable model — not just raw accuracy.

**In this case:**

**Ridge Regression** strikes the best balance between **accuracy, simplicity, and generalization.**