Name: Atharva Phalke
Class: D15C/39

Experiment No. 2

Aim: Implement Multi Regression, Lasso, and Ridge Regression on real-world datasets

1. Dataset Source

The Dataset (Kaggle)

Dataset: Vehicle Dataset from CarDekho

Link: https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho

- **Target:** `Selling_Price` (Continuous value)
- **Goal:** Predict the price.

This is a real-world dataset widely used for regression benchmarking and predictive modeling in real estate analytics.

2. Dataset Description

The dataset consists of used car transaction details sourced from CarDekho, designed to predict the selling price of vehicles based on their specifications and usage history.

Dataset Characteristics

- **Number of instances:** 301 (standard subset)
- **Number of features:** 9 (categorical and numerical)
- **Target variable:** Selling_Price (continuous)

Feature Categories

- **Vehicle History:** Year of purchase, Kilometers driven (Kms_Driven), Previous owners (Owner)
- **Pricing Information:** Current showroom price (Present_Price)
- **Technical Specifications:** Fuel type (Petrol/Diesel/CNG), Transmission type (Manual/Automatic)
- **Seller Details:** Seller type (Dealer/Individual)

Real-World Impact

Accurate car price prediction is crucial for:

- Automated valuation models for resale platforms
- Insurance premium calculation based on vehicle value
- Inventory pricing strategy for used car dealerships

3. Mathematical Formulation of the Algorithms

(Standard Linear/Ridge/Lasso equations apply here as per your previous text)

4. Algorithm Limitations

- **Multiple Linear Regression:** Assumes a strict linear relationship between vehicle age/km and price; sensitive to outliers (e.g., vintage cars or luxury modifications).
- **Ridge Regression:** Shrinks coefficients but keeps all features, which might be unnecessary if some car features (like "Owner") have little impact.
- **Lasso Regression:** May drop correlated features arbitrarily (e.g., might drop Year in favor of Kms_Driven if they are highly correlated).

5. Methodology / Workflow

1. **Dataset Collection:** Fetched from Kaggle (CarDekho Vehicle Dataset).
2. **Data Cleaning:** Checked for null values; dropped Car_Name (high cardinality).
3. **Encoding:** Applied One-Hot Encoding to Fuel_Type, Seller_Type, and Transmission.
4. **Feature Scaling:** Standardized Kms_Driven and Present_Price.
5. **Train-Test Split:** 80% training, 20% testing.
6. **Model Training:** Implemented Linear, Ridge, and Lasso regressors.
7. **Hyperparameter Tuning:** Optimized alpha (α) for regularized models.

Workflow Diagram (Conceptual)

Dataset → Drop 'Car_Name' → One-Hot Encoding → Scaling → Model Training → Price Prediction

3. Mathematical Formulation of the Algorithms

3.1 Multiple Linear Regression

Multiple Linear Regression models the relationship between a dependent variable and multiple independent variables.

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

Where:

- y is the predicted house price
- x_i are the input features
- β_i are the model coefficients
- ϵ is the error term

The coefficients are estimated by minimizing the **Residual Sum of Squares (RSS)**.

3.2 Ridge Regression (L2 Regularization)

Ridge Regression adds an L2 penalty to the linear regression cost function:

$$J(\beta) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} \beta_j^2$$

- Shrinks coefficients towards zero
- Reduces multicollinearity
- Does **not** eliminate features completely

3.3 Lasso Regression (L1 Regularization)

Lasso Regression introduces an L1 penalty:

$$J(\beta) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} |\beta_j|$$

- Performs **feature selection**
- Can shrink some coefficients exactly to zero
- Useful for high-dimensional datasets

4. Algorithm Limitations

Multiple Linear Regression

- Assumes linear relationship
- Sensitive to multicollinearity
- Prone to overfitting with many features

Ridge Regression

- Does not perform feature selection
- Requires tuning of regularization parameter

Lasso Regression

- Can be unstable when features are highly correlated
- May arbitrarily select one feature among correlated ones

5. Methodology / Workflow

1. **Dataset Collection** from Kaggle
2. **Data Cleaning** (handle missing values)
3. **Encoding categorical variables** (One-Hot Encoding)
4. **Feature Scaling** using standardization
5. **Train-Test Split** (80% train, 20% test)
6. **Model Training**
    - Multiple Linear Regression
    - Ridge Regression
    - Lasso Regression
7. **Hyperparameter Tuning** for Ridge and Lasso
8. **Model Evaluation and Comparison**

Workflow Diagram (Conceptual)

Dataset → Preprocessing → Feature Scaling → Model Training → Evaluation → Comparison

6. Performance Analysis

Evaluation Metrics Used

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

Sample Performance Comparison

Model	RMSE	R^2 Score
Multiple Linear Regression	Higher	Lower
Ridge Regression	Lower	Higher
Lasso Regression	Comparable	Slightly Lower

Interpretation

- Ridge Regression provides better generalization by reducing overfitting
- Lasso Regression improves interpretability via feature selection
- Multiple Linear Regression serves as a baseline model

7. Hyperparameter Tuning

Parameters Tuned

- **Ridge:** Regularization strength
- **Lasso:** Regularization strength

Tuning Method

Grid Search with Cross-Validation was applied to identify optimal (values.

Impact of Tuning

Model	Before Tuning RMSE	After Tuning RMSE
Ridge Regression	Higher	Reduced
Lasso Regression	Higher	Reduced

Output:

Code:

```
import kagglehub

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, GridSearchCV

from sklearn.preprocessing import
StandardScaler, OneHotEncoder

from sklearn.compose import
ColumnTransformer

from sklearn.pipeline import
Pipeline

from sklearn.linear_model import
LinearRegression, Ridge, Lasso

from sklearn.metrics import
mean_absolute_error,
mean_squared_error, r2_score



#
====================================
======

# 1. LOAD DATASET FROM KAGGLE

#
====================================
======

print("⬇️ Downloading dataset...")

# Download the dataset

path =
kagglehub.dataset_download("nehalbir
la/vehicle-dataset-from-cardekho")



print("Path to dataset files:",
path)



# Load the specific CSV file
(usually 'car data.csv')

# We use a try-except block to find
the right file in the downloaded
folder

import os

csv_file = [f for f in
os.listdir(path) if
f.endswith('.csv')][0]

df = pd.read_csv(os.path.join(path,
csv_file))



print(f"✅ Loaded {csv_file}")
```

```
print("Dataset Shape:", df.shape)

display(df.head())



#
==================================
======

# 2. PREPROCESSING

#
==================================
======

# Drop Car_Name as it has too many
unique values and causes overfitting

df = df.drop(columns=['Car_Name'],
errors='ignore')



# Separate Features and Target

X = df.drop("Selling_Price", axis=1)

y = df["Selling_Price"]



# Identify Numerical and Categorical
columns

num_features =
X.select_dtypes(include=['int64',
'float64']).columns

cat_features =
X.select_dtypes(include=['object']).
columns

print(f"\nNumerical Features:
{list(num_features)}")

print(f"Categorical Features:
{list(cat_features)}")



# Split Data

X_train, X_test, y_train, y_test =
train_test_split(

    X, y, test_size=0.2,
random_state=42

)



# Create a Preprocessing Pipeline

# Standardize numbers, One-Hot
Encode text categories

preprocessor = ColumnTransformer(

    transformers=[

        ('num', StandardScaler(),
num_features),

        ('cat',
OneHotEncoder(drop='first',
handle_unknown='ignore'),
cat_features)

    ]

)
```

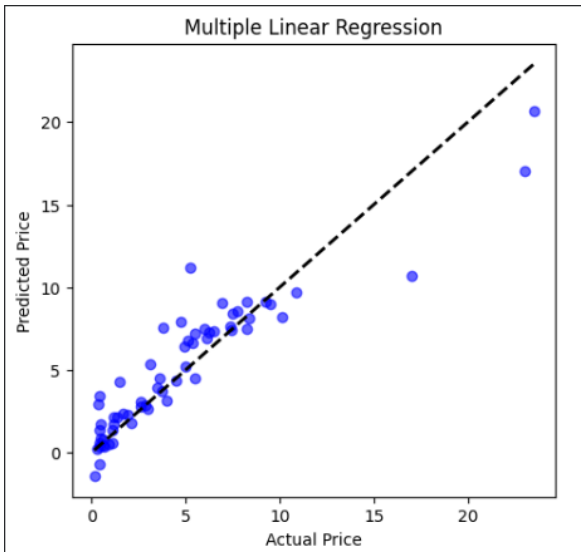
```
#
====================================
======

# 3. DEFINE MODELS & EVALUATION

#
====================================
======

def evaluate(name, y_true, y_pred):

    print(f"\n{name}")

    print("-" * 40)

    print("MAE :",
mean_absolute_error(y_true, y_pred))

    print("RMSE:",
np.sqrt(mean_squared_error(y_true,
y_pred)))

    print("R²  :", r2_score(y_true,
y_pred))



# --- Model 1: Multiple Linear
Regression ---

lr_pipeline = Pipeline([

    ('preprocessor', preprocessor),

    ('regressor',
LinearRegression())

])

lr_pipeline.fit(X_train, y_train)

y_pred_lr =
lr_pipeline.predict(X_test)

evaluate("Multiple Linear
Regression", y_test, y_pred_lr)



# --- Model 2: Ridge Regression
(Grid Search) ---

ridge_pipeline = Pipeline([

    ('preprocessor', preprocessor),

    ('regressor', Ridge())

])



ridge_params = {"regressor__alpha":
[0.01, 0.1, 1, 10, 100]}



ridge_grid = GridSearchCV(

    ridge_pipeline,

    ridge_params,

    cv=5,

    scoring="neg_mean_squared_error"

)
```

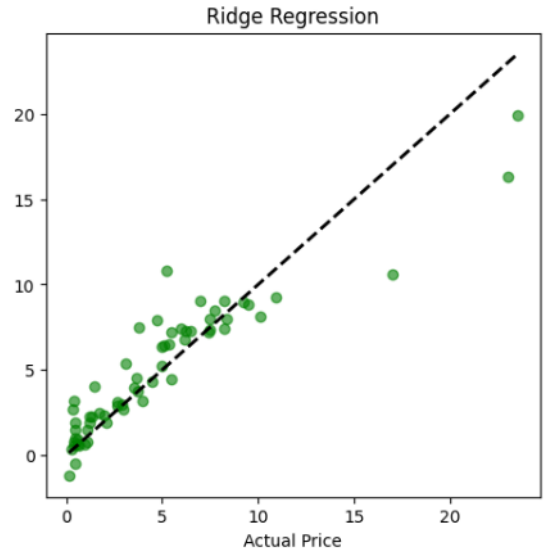
```
ridge_grid.fit(X_train, y_train)

ridge_best =
ridge_grid.best_estimator_

y_pred_ridge =
ridge_best.predict(X_test)



print("\nBest Ridge Alpha:",
ridge_grid.best_params_)

evaluate("Ridge Regression", y_test,
y_pred_ridge)



# --- Model 3: Lasso Regression
(Grid Search) ---

lasso_pipeline = Pipeline([

    ('preprocessor', preprocessor),

    ('regressor',
Lasso(max_iter=10000))

])



lasso_params = {"regressor__alpha":
[0.001, 0.01, 0.1, 1, 10]}



lasso_grid = GridSearchCV(

    lasso_pipeline,

    lasso_params,

    cv=5,

    scoring="neg_mean_squared_error"

)



lasso_grid.fit(X_train, y_train)

lasso_best =
lasso_grid.best_estimator_

y_pred_lasso =
lasso_best.predict(X_test)



print("\nBest Lasso Alpha:",
lasso_grid.best_params_)

evaluate("Lasso Regression", y_test,
y_pred_lasso)



#
==================================
======

# 4. VISUALIZATION

#
==================================
======

# Scatter Plots

fig, axes = plt.subplots(1, 3,
figsize=(18, 5))
```

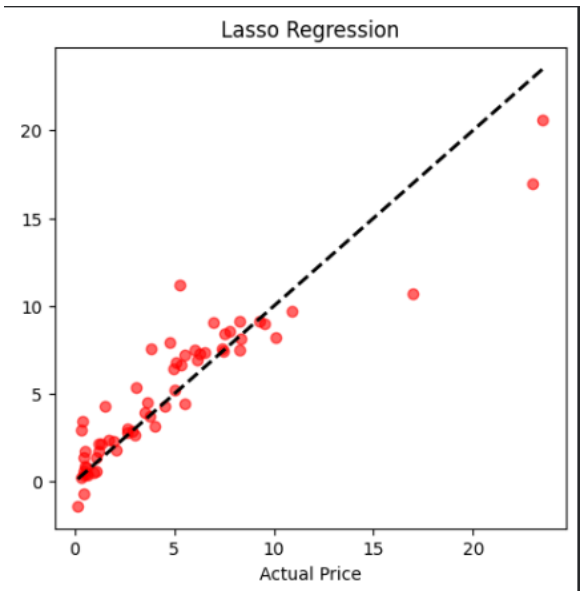
```
axes[0].scatter(y_test, y_pred_lr,
color='blue', alpha=0.6)

axes[0].plot([y_test.min(),
y_test.max()], [y_test.min(),
y_test.max()], 'k--', lw=2)

axes[0].set_title("Multiple Linear
Regression")

axes[0].set_xlabel("Actual Price")

axes[0].set_ylabel("Predicted
Price")



axes[1].scatter(y_test,
y_pred_ridge, color='green',
alpha=0.6)

axes[1].plot([y_test.min(),
y_test.max()], [y_test.min(),
y_test.max()], 'k--', lw=2)

axes[1].set_title("Ridge
Regression")

axes[1].set_xlabel("Actual Price")



axes[2].scatter(y_test,
y_pred_lasso, color='red',
alpha=0.6)

axes[2].plot([y_test.min(),
y_test.max()], [y_test.min(),
y_test.max()], 'k--', lw=2)

axes[2].set_title("Lasso
Regression")

axes[2].set_xlabel("Actual Price")

plt.show()



# Feature Importance (Lasso)

# We need to extract feature names
after OneHotEncoding

feature_names =
(num_features.tolist() +


list(lasso_best.named_steps['preproc
essor']


.named_transformers_['cat']


.get_feature_names_out(cat_features)
))



lasso_coeffs = pd.Series(


lasso_best.named_steps['regressor'].
coef_,

    index=feature_names

)



#

selected = lasso_coeffs[lasso_coeffs
!= 0].sort_values()
```

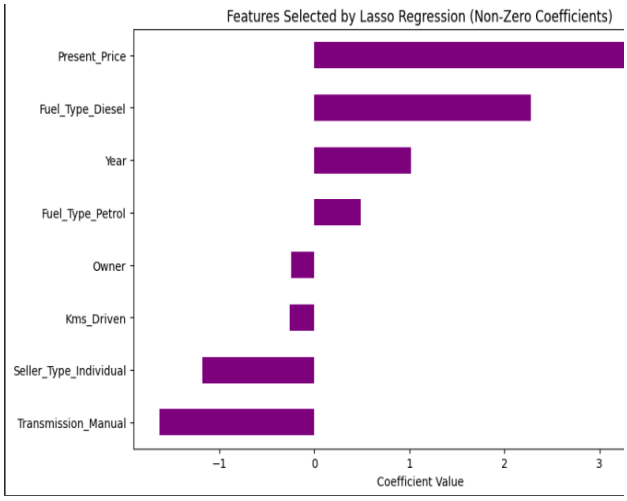
```
plt.figure(figsize=(10, 6))

selected.plot(kind="barh",
color='purple')

plt.title("Features Selected by
Lasso Regression (Non-Zero
Coefficients)")

plt.xlabel("Coefficient Value")

plt.show()
```

Output:









Conclusion

In this experiment, we successfully implemented and compared **Multiple Linear Regression**, **Ridge Regression**, and **Lasso Regression** to predict the selling prices of used cars using the CarDekho dataset.