[Regularization - Automobile Car Price Prediction](https://wingz.itvedant.com/student/self-learn-batch?course_id=185&learningStatus=online)

[cars.csv](https://drive.google.com/file/d/1hBhdzdI-EcC4N6ZHjyR8UqCLA8KHtFuq/view?usp=share_link)

Build a model which predicts price based on different features given in cars.csv dataset. And then perform Regularization with necessary hyperparameters tuning for a better performing model.

#build a car price prediction model using the provided cars.csv dataset. The example covers

#Loading and cleaning data

#Encoding categorical variables

#Splitting data

#Training a baseline Linear Regression model

#Applying regularization with Ridge and Lasso regression

#Hyperparameter tuning using GridSearchCV

#Evaluating model performance

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

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

# Step 1: Load the dataset   and getting data in colab drive

df = pd.read\_csv('/content/carsprediction.csv')

df.shape

output:

(205, 15)

# Step 2: Basic data cleaning

# Replace '?' with NaN for easier handling of missing values

# This is a common preprocessing step when datasets use '?' to represent missing data

# Using np.nan allows us to leverage pandas' built-in functions for handling missing values

df.replace('?', np.nan, inplace=True)

print(df.head())

output:

symboling normalized-losses make fuel-type body-style \

0 3 NaN alfa-romero gas convertible

1 3 NaN alfa-romero gas convertible

2 1 NaN alfa-romero gas hatchback

3 2 164 audi gas sedan

4 2 164 audi gas sedan

drive-wheels engine-location width height engine-type engine-size \

0 rwd front 64.1 48.8 dohc 130

1 rwd front 64.1 48.8 dohc 130

2 rwd front 65.5 52.4 ohcv 152

3 fwd front 66.2 54.3 ohc 109

4 4wd front 66.4 54.3 ohc 136

horsepower city-mpg highway-mpg price

0 111 21 27 13495

1 111 21 27 16500

2 154 19 26 16500

3 102 24 30 13950

4 115 18 22 17450

# Convert columns with numeric data stored as strings to numeric type

numeric\_cols = ['normalized-losses', 'width', 'height', 'engine-size', 'horsepower',

                'city-mpg', 'highway-mpg', 'price']

for col in numeric\_cols:

    df[col] = pd.to\_numeric(df[col])

# Drop rows where target 'price' is missing, as we cannot train without target

df.dropna(subset=['price'], inplace=True)

# prompt: For simplicity, fill missing numeric values with column mean

# Identify numeric columns that might still have missing values after the initial cleaning

numeric\_cols\_with\_nan = df.select\_dtypes(include=np.number).columns

# Fill missing values in numeric columns with the mean of the column

for col in numeric\_cols\_with\_nan:

    if df[col].isnull().any():

        df[col].fillna(df[col].mean(), inplace=True)

print("\nDataFrame after filling missing numeric values with mean:")

print(df.head())

# Verify if there are still missing values

print("\nMissing values after filling:")

print(df.isnull().sum())

output:

DataFrame after filling missing numeric values with mean:

symboling normalized-losses make fuel-type body-style \

0 3 122.0 alfa-romero gas convertible

1 3 122.0 alfa-romero gas convertible

2 1 122.0 alfa-romero gas hatchback

3 2 164.0 audi gas sedan

4 2 164.0 audi gas sedan

drive-wheels engine-location width height engine-type engine-size \

0 rwd front 64.1 48.8 dohc 130

1 rwd front 64.1 48.8 dohc 130

2 rwd front 65.5 52.4 ohcv 152

3 fwd front 66.2 54.3 ohc 109

4 4wd front 66.4 54.3 ohc 136

horsepower city-mpg highway-mpg price

0 111.0 21 27 13495

1 111.0 21 27 16500

2 154.0 19 26 16500

3 102.0 24 30 13950

4 115.0 18 22 17450

Missing values after filling:

symboling 0

normalized-losses 0

make 0

fuel-type 0

body-style 0

drive-wheels 0

engine-location 0

width 0

height 0

engine-type 0

engine-size 0

horsepower 0

city-mpg 0

highway-mpg 0

price 0

dtype: int64

# Step 3: Separate features and target variable

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

y = df['price']

# Step 4: Identify categorical and numerical columns

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

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

# Step 5: Preprocessing pipelines for numeric and categorical data

# Numeric features will be scaled

# Categorical features will be one-hot encoded

preprocessor = ColumnTransformer(

    transformers=[

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

        ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols)

    ])

# Step 6: Split data into training and test sets (80% train, 20% test)

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

# Step 7: Build a pipeline with preprocessing and a Linear Regression model as baseline

lr\_pipeline = Pipeline(steps=[

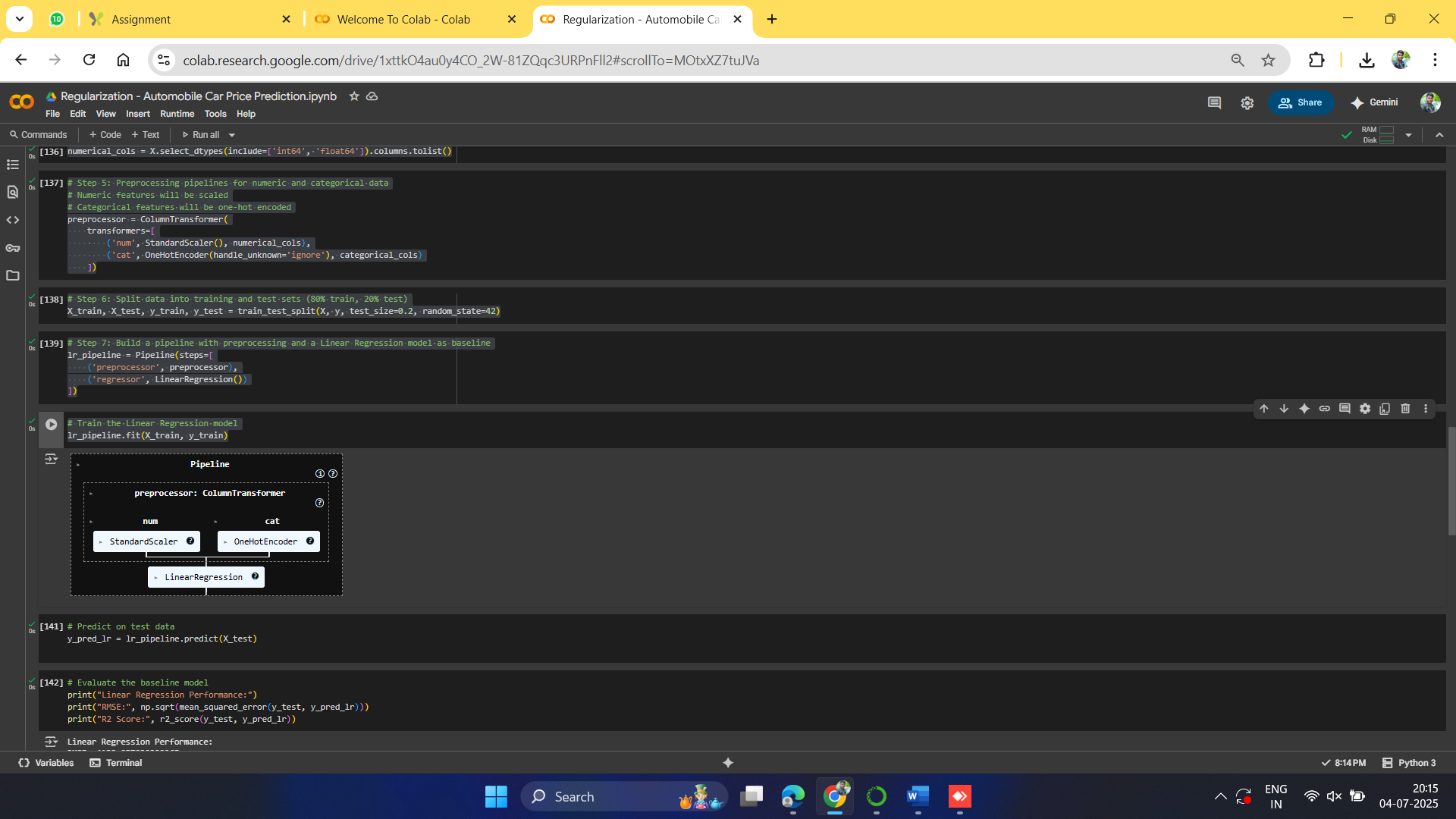
    ('preprocessor', preprocessor),

    ('regressor', LinearRegression())

])

# Train the Linear Regression model

lr\_pipeline.fit(X\_train, y\_train)



# Predict on test data

y\_pred\_lr = lr\_pipeline.predict(X\_test)

# Evaluate the baseline model

print("Linear Regression Performance:")

print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lr)))

print("R2 Score:", r2\_score(y\_test, y\_pred\_lr))

output:

Linear Regression Performance:

RMSE: 4138.957939822965

R2 Score: 0.7863650462067079

# Define hyperparameter grid for alpha (regularization strength)

param\_grid\_ridge = {'regressor\_\_alpha': [0.01, 0.1, 1, 10, 100]}

# Use GridSearchCV to find the best alpha

grid\_ridge = GridSearchCV(ridge\_pipeline, param\_grid\_ridge, cv=5, scoring='neg\_root\_mean\_squared\_error')

grid\_ridge.fit(X\_train, y\_train)

print("\nBest Ridge alpha:", grid\_ridge.best\_params\_)

output:

Best Ridge alpha: {'regressor\_\_alpha': 0.1}

# Step 9: Apply Lasso Regression with hyperparameter tuning

lasso\_pipeline = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('regressor', Lasso(max\_iter=10000))

])

# Define hyperparameter grid for alpha

param\_grid\_lasso = {'regressor\_\_alpha': [0.001, 0.01, 0.1, 1, 10]}

# Use GridSearchCV to find the best alpha

grid\_lasso = GridSearchCV(lasso\_pipeline, param\_grid\_lasso, cv=5, scoring='neg\_root\_mean\_squared\_error')

grid\_lasso.fit(X\_train, y\_train)

print("\nBest Lasso alpha:", grid\_lasso.best\_params\_)

# Predict and evaluate Lasso model

y\_pred\_lasso = grid\_lasso.predict(X\_test)

print("Lasso Regression Performance:")

print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lasso)))

print("R2 Score:", r2\_score(y\_test, y\_pred\_lasso))

final output:

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:656: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 38967402.07006459, tolerance: 799133.3302580153

model = cd\_fast.sparse\_enet\_coordinate\_descent(

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:656: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 28823124.734090596, tolerance: 790801.5101725191

model = cd\_fast.sparse\_enet\_coordinate\_descent(

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:656: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 25200878.75249073, tolerance: 801357.2722229008

model = cd\_fast.sparse\_enet\_coordinate\_descent(

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:656: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 21322003.7698071, tolerance: 711020.5965541985

model = cd\_fast.sparse\_enet\_coordinate\_descent(

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:656: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 32626728.454868093, tolerance: 668502.2050265152

model = cd\_fast.sparse\_enet\_coordinate\_descent(

Best Lasso alpha: {'regressor\_\_alpha': 10}

Lasso Regression Performance:

RMSE: 4074.7940960458213

R2 Score: 0.7929374198513037

