Faculty of Engineering & Technology

Department of Electrical & Computer Engineering

ENCS5341 – Machine Learning and Data Since

First Semester 2024/2025

Assignment #2

Prepared by

Jalila Muadi – 1201611

Section No. 1

Instructor: Dr. Yazan Abu Farha

BIRZEIT

November 2024

Abstract

This project explores the use of machine learning techniques to predict car prices using a dataset sourced from YallaMotors. The study focuses on applying and comparing linear regression models, including closed-form and gradient descent methods, as well as regularization techniques like LASSO and Ridge. Additionally, nonlinear models such as Polynomial Regression and Gaussian Kernel (RBF) are evaluated to capture complex relationships within the data. Key preprocessing steps, including handling missing values, encoding categorical features, and standardizing numerical data, were meticulously performed to prepare the dataset. The models were evaluated using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared, with hyperparameter tuning and feature selection enhancing the overall performance. This Assignment highlights the significance of employing advanced regression techniques and proper model selection to achieve robust predictions.

Table of Contents

Abstract ..... **I**

List of Figures ..... **III**

Introduction **1**

Dataset Description and Preprocessing **2**

Handling missing values 4

Encoding categorical features 11

Normalizing/standardizing numerical features 11

Splitting into training, validation, and test sets 12

Distribution of the target variable 13

Building Regression Models **14**

1. Linear Models: 14

1.1. Closed-Form Linear Regression Model 14

1.2. Gradient Descent Linear Regression Model 14

2. Regularization Techniques 16

2.1. LASSO (L1) Regularization: 16

2.2. Ridge (L2) Regularization: 17

Discussion and Conclusion 18

3. Nonlinear models 20

3.1. Polynomial Regression 20

3.2. Gaussian Kernel (RBF) 22

Model Evaluation on Test Set **24**

Feature Selection Using Forward Selection **26**

Discussion **28**

Conclusion **29**

Reference 30

List of Figures

Figure 1: Dataset Information 3
Figure 2: Missing Values for each feature 3
Figure 3: Missing Value Summary After Process 4
Figure 4: Before replacing with mean price for each brand 5
Figure 5: After replacing with mean price for each brand 5
Figure 6: Remaining Price Missing Values 5
Figure 7: Engine Capacity Boxplot Before Handling Outlier 6
Figure 8: Engine Capacity Boxplot After Handling Outlier 6
Figure 9: Engine Capacity Boxplot After Handling all Missing Values with Median Engine Capacity for Each Brand 6
Figure 10: Address Missing Values for Electric Vehicles (EVs) 7
Figure 11: horse_power Boxplot Before Handle Outlier 8
Figure 12: horse_power Boxplot After Handle Outlier 8
Figure 13: Before apply the cleaning 8
Figure 14: After apply the cleaning 8
Figure 15: top_speed Boxplot Before Handle Outlier 9
Figure 16: top_speed Boxplot After Handle Outlier 9
Figure 17: Missing Seats 10
Figure 18: Data Form After Encoding 11
Figure 19: Standardized Cars Dataset 12
Figure 20: Distribution of Car Prices in USD 13
Figure 21: Box Plot of Car Prices 13
Figure 22: performance metrics of the Gradient Descent and Closed-Form Solution 14
Figure 23: Weight Comparison Table 15
Figure 24: LASSO Regularization Results for Different Value for λ 16
Figure 25: LASSO Convergence (Lambda = 0.01) 16
Figure 26: Ridge Regularization Results 17

Figure 27: Ridge Regression Performance Across Lambda Values 18
Figure 28: Performance Metrics of Polynomial Regression Models (Degrees 2 to 10) 20
Figure 29: Polynomial Regression: MSE vs. Degree 21
Figure 30: Polynomial Regression: R^2 vs. Degree 21
Figure 31: Performance Metrics for Train and evaluate RBF Kernel Regression with Ridge Regression 22
Figure 32: MSE for RBF Kernel Regression (Gamma vs. Alpha) 23
Figure 33: All Models Performance on Validation set 24
Figure 34: Best Model 25
Figure 35: Test Set Performance for Best Polynomial Regression Model 25
Figure 36: Selected Features 26
Figure 37: Performance with Selected Features 27
Figure 38: Performance with All Features 27

Introduction

In today’s dynamic automotive market, accurately predicting car prices is essential for buyers, sellers, and analysts alike. This Assignment harnesses machine learning techniques to forecast car prices using a dataset from **Yalla Motors**, which includes diverse features such as car make, model, year, mileage, engine capacity, and horsepower. By analyzing these attributes, the Assignment seeks to uncover their relationships with pricing and provide actionable insights.

The primary objective is to implement, evaluate, and compare various regression models both linear (Linear Regression, LASSO, and Ridge) and nonlinear (Polynomial Regression and Gaussian Kernel) to identify the most accurate approach for price prediction. Regularization techniques like LASSO and Ridge are utilized to mitigate overfitting, while feature selection via forward selection ensures optimal model performance. Furthermore, hyperparameter tuning is conducted to fine-tune each model for accuracy and robustness.

This report details the methodology employed, encompassing data preprocessing, exploratory data analysis (EDA), model development, and performance evaluation. Metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared are used to assess model performance comprehensively. By examining the relationships between car features and pricing, this study underscores the predictive power of regression analysis and the importance of advanced techniques in building reliable models.

Dataset Description and Preprocessing

The dataset was sourced from **Yalla Motors**. It was scraped using Python and the Requests-HTML library, resulting in a comprehensive dataset with approximately 6,310 rows and 9 columns (features). The dataset contains the following features, which provide detailed information about the cars:

Feature	Description
Car Name	The name of the car, including its make and model.
Price	The selling price of the car (target variable).
Engine Capacity	The engine size, typically measured in liters or cubic centimeters (cc).
Cylinder	The number of cylinders in the car's engine, reflecting its power.
Horse Power	The horsepower output of the car, indicating its engine performance.
Top Speed	The maximum speed the car can achieve, measured in km/h.
Seats	The number of seats available in the car.
Brand	The manufacturer or brand of the car (Toyota, BMW).
Country	The country where the car is listed for sale.

The **Price** column serves as the target variable for regression analysis. It represents the monetary value of the car and is critical for developing predictive models. However, several potential issues may arise:

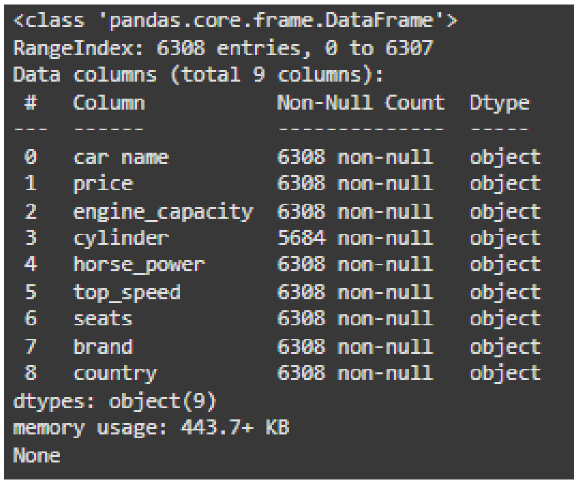
1. **Inconsistent Currencies**: Prices may be listed in various currencies depending on the country where the car is sold. This inconsistency must be addressed by standardizing all prices to a common currency (e.g., USD) to ensure uniformity.

2. **Outliers**: High-end luxury cars or rare vehicles might have extreme price values that could skew the model's performance. Outlier detection and handling will be necessary.

3. **Missing Values**: Missing or incomplete price entries could impact the model training and need to be imputed or removed.

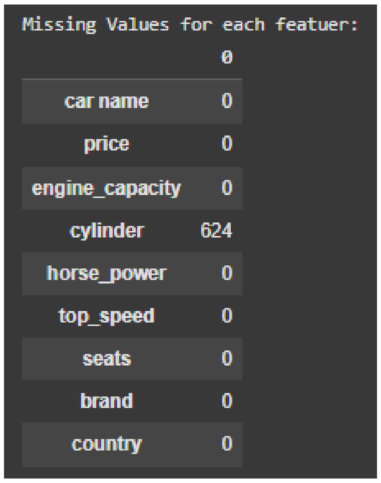
By carefully addressing these challenges, the dataset can be effectively utilized for predictive modeling, enabling insights into the relationships between car attributes and their market prices.

All columns in the dataset are currently of type Object, which indicates potential issues with numeric columns stored as strings. Additionally, some columns contain outliers or invalid values, including price, engine_capacity, cylinder, horse_power, top_speed, and seats. Initially, the cylinder column was identified to have missing values, with 624 entries requiring cleaning or transformation.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6308 entries, 0 to 6307
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   car name         6308 non-null   object
 1   price            6308 non-null   object
 2   engine_capacity  6308 non-null   object
 3   cylinder         5684 non-null   object
 4   horse_power      6308 non-null   object
 5   top_speed        6308 non-null   object
 6   seats            6308 non-null   object
 7   brand            6308 non-null   object
 8   country          6308 non-null   object
dtypes: object(9)
memory usage: 443.7+ KB
None
```

Figure 1: Dataset Information



Missing Values for each featuer:

	0
car name	0
price	0
engine_capacity	0
cylinder	624
horse_power	0
top_speed	0
seats	0
brand	0
country	0

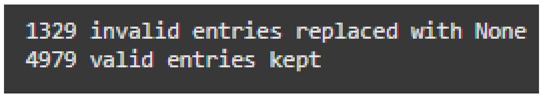
Figure 2: Missing Values for each feature

The dataset contains 6,308 entries across nine columns, as shown in Figure 1 and the count of missing values for each feature is detailed in Figure 2. One feature, cylinder, was found to have 624 missing values. The remaining features do not have missing values but may require cleaning or transformation for consistency.

The price column initially contained several invalid entries such as "Orange burst Metallic," "TBD," and other non-numerical strings. These entries are not useful for regression analysis as they do not represent numerical values. To address this issue, the following steps were taken:

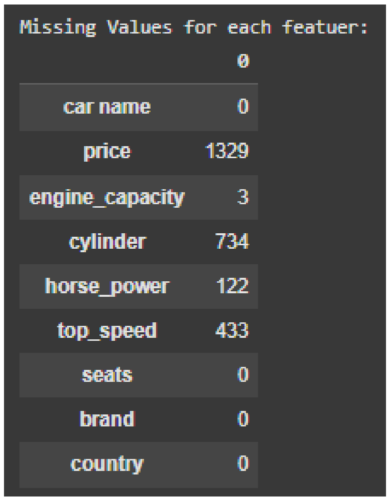
- Entries with a valid currency format (e.g., "SAR 76,545") were retained.
- All other invalid entries were replaced with None.

After the cleaning process:



```
1329 invalid entries replaced with None
4979 valid entries kept
```

The cleaning process applied to the **price** column was also applied to the following columns: **engine_capacity**, **cylinder**, **horse_power**, **top_speed.**



```
Missing Values for each featuer:
                      0
       car name       0
          price    1329
engine_capacity       3
       cylinder     734
    horse_power     122
      top_speed     433
          seats       0
          brand       0
        country       0
```

Figure 3: Missing Value Summary After Process

The **seats** column contains missing values and float numbers, which are unusual for this feature. However, these issues will be handled later.

Handling missing values

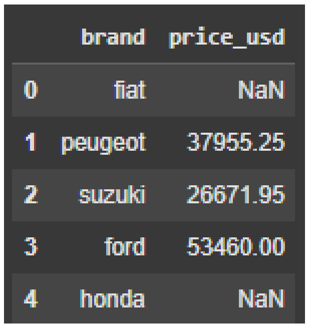
In this section, we will handle the missing values across all features.

Handling Missing Values in the Price Column

First, I analyzed the price column and identified the currency types in the dataset: AED, QAR, KWD, OMR, BHD, SAR, and EGP. To standardize the values and avoid discrepancies, all listed currencies were converted to USD using the appropriate exchange rates:

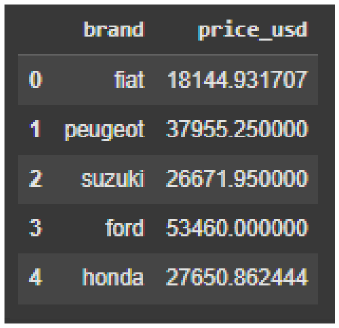
Currency	Exchange Rate to USD
AED	0.27
QAR	0.27
KWD	3.27
OMR	2.60
BHD	2.65
SAR	0.27
EGP	0.032

The converted values replaced the original prices, and the column renamed from price to **price_usd**. After converting the prices, missing values in the price_usd column were replaced with the mean price for each brand. This approach ensured that the replacement values were representative of typical pricing trends for each brand.



	brand	price_usd
0	fiat	NaN
1	peugeot	37955.25
2	suzuki	26671.95
3	ford	53460.00
4	honda	NaN

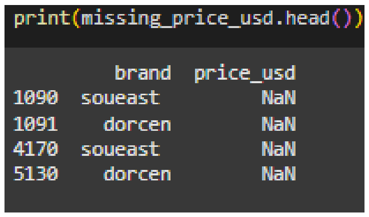
Figure 4: Before replacing with mean price for each brand



	brand	price_usd
0	fiat	18144.931707
1	peugeot	37955.250000
2	suzuki	26671.950000
3	ford	53460.000000
4	honda	27650.862444

Figure 5: After replacing with mean price for each brand

After imputing missing values, 4 records still lacked prices. This occurred because the brands associated with these records (Soueast and Dorcen) did not have sufficient data in the dataset to compute a meaningful mean price.



```
print(missing_price_usd.head())

        brand  price_usd
1090  soueast        NaN
1091   dorcen        NaN
4170  soueast        NaN
5130   dorcen        NaN
```

Figure 6: Remaining Price Missing Values

The 4 records with missing prices were removed from the dataset, as they represented an insignificant portion of the data and could not be reliably imputed.

Handling Outlier & Missing Values in the Engine Capacity Column

Upon visualizing the engine capacity column using a boxplot, it became evident that the column contained numerous outlier values, with many exceeding typical engine capacities. For most vehicles, engine capacities usually fall within the range of **0 to 8 liters**, while electric vehicles (EVs), which do not have traditional internal combustion engines, are represented with an **engine capacity of 0**, which is considered valid.

i note from the data set, several values exceeded typical ranges, with some values such as 3995 and 3982 representing clear outliers. Before considering the valid range of 0 to 8 liters, values greater than 1000 were transformed into a fractional form to approximate realistic engine capacities:

For values greater than 1000, the engine capacity was divided by 1000 to convert the value into fractional form. For example, 3995 was transformed to 3.995 and approximated to 4. Values smaller than 1000 were retained without modification.

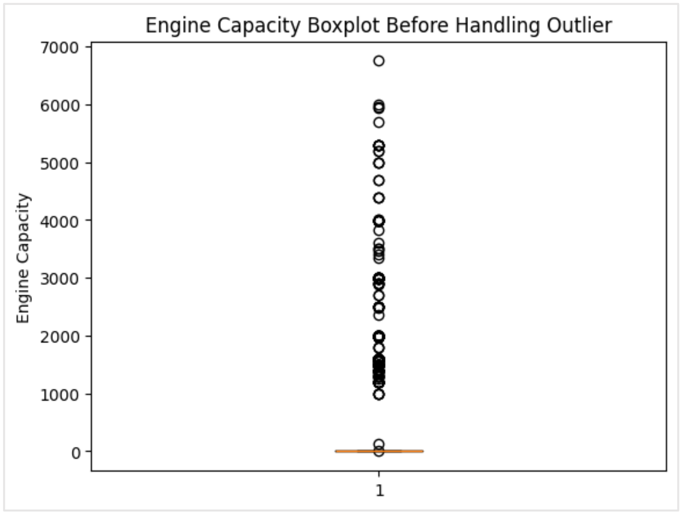


Figure 7: Engine Capacity Boxplot Before Handling Outlier

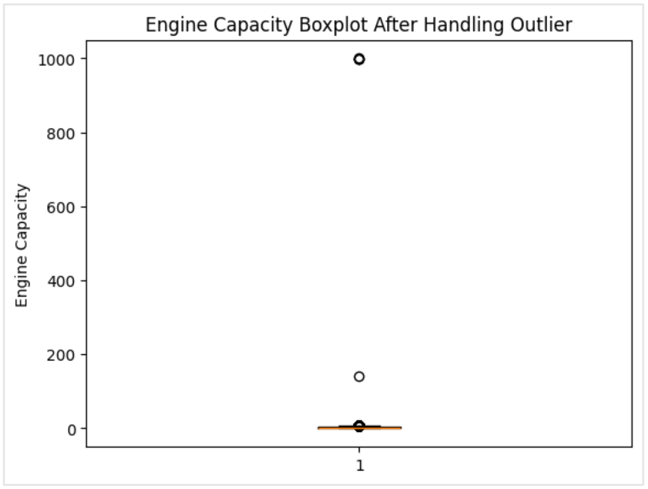


Figure 8: Engine Capacity Boxplot After Handling Outlier

Then any value in the engine_capacity column that exceeded 8 liters or fell below 0 was treated as an outlier. These values were replaced with NaN to mark them as missing data.

To handle these missing values including the outliers replaced with NaN, I replaced them with the **median** engine capacity for each brand. This method ensures that the imputed values are representative of the typical engine capacities within each brand.

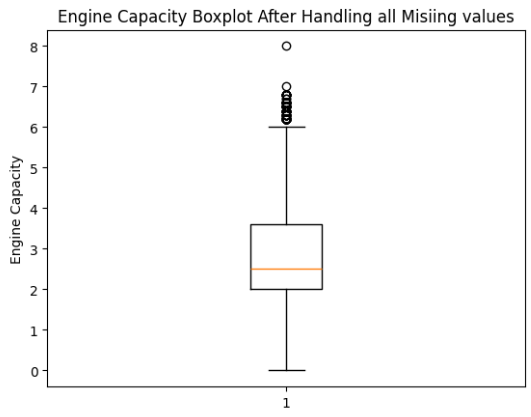


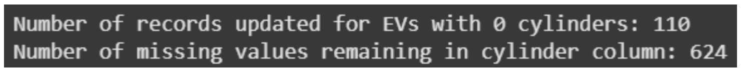
Figure 9: Engine Capacity Boxplot After Handling all Missing Values with Median Engine Capacity for Each Brand

Handling Missing Values in the cylinder Column

In this step, we addressed missing values in the cylinder column by implementing a logical and systematic approach to ensure data consistency and accuracy.

First, grouping by car name to Replace Missing Values with the Median. Missing values in the cylinder column were first handled by grouping the data based on the car name and replacing the missing entries with the median number of cylinders for each specific car model. This approach ensures that the imputation of missing values aligns with the general characteristics of the car model.

Then, electric cars (EVs) do not have cylinders because they do not use internal combustion engines. Instead, they rely on electric motors. Therefore, for any record where the **engine_capacity** is 0, the corresponding **cylinder** value (missing) was set to 0. This step ensures that EVs are correctly identified and handled in the dataset.



```
Number of records updated for EVs with 0 cylinders: 110
Number of missing values remaining in cylinder column: 624
```

Figure 10: Address Missing Values for Electric Vehicles (EVs)

After handling EVs, there were still 624 missing values in the cylinder column. To address these, the data was further grouped by the brand of the car, and missing cylinder values were replaced with the median number of cylinders for each brand.

Handling Missing Values in the horse_power Column

In this section, missing and outlier values in the horsepower column were addressed. The process involved grouping by relevant attributes and defining valid ranges for horsepower values.

First, missing values in the horse_power column were initially replaced with the median horsepower value for each car model (car name). Then I analyzed the horsepower column using a box plot. The plot revealed the presence of outliers. Outliers were defined as i understand:

- horse_power < 60: Extremely low values unlikely for a car engine.
- horse_power > 2500: Unrealistically high values for most vehicles.

These outlier values were replaced with NaN to mark them as missing for imputation.

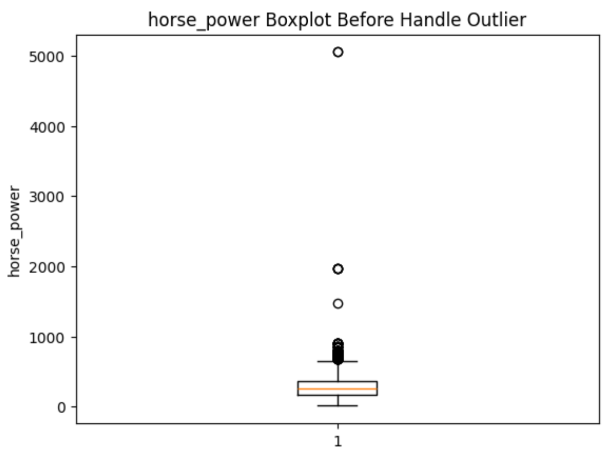


Figure 11: horse_power Boxplot Before Handle Outlier

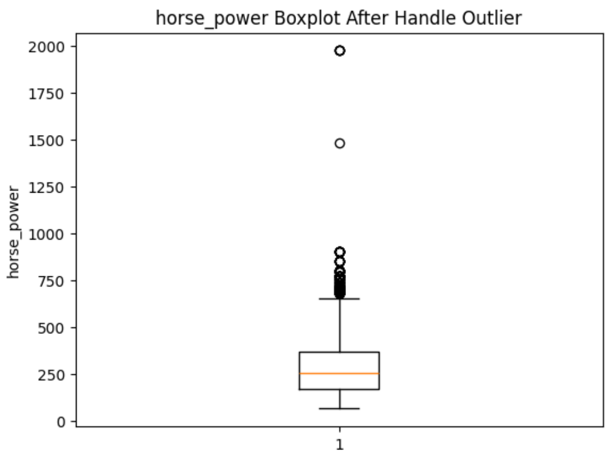
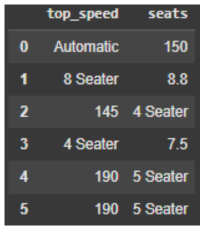


Figure 12: horse_power Boxplot After Handle Outlier

After handling the outliers, for the remaining missing values, the horsepower column was grouped by brand, and the missing values were replaced with the median horsepower for each brand.

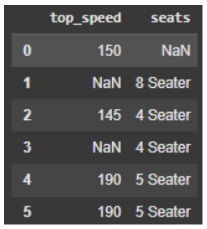
Handling Missing Values in the top_speed Column

When exploring the dataset, I identified several inconsistencies in the top_speed column. Many records in the column contained non-numeric strings instead of numeric values. Some strings in the top_speed column referred to the number of seats (e.g., "8-Seater"), which conflicted with the seats column that already represented the number of seats.



	top_speed	seats
0	Automatic	150
1	8 Seater	8.8
2	145	4 Seater
3	4 Seater	7.5
4	190	5 Seater
5	190	5 Seater

Figure 13: Before apply the cleaning



	top_speed	seats
0	150	NaN
1	NaN	8 Seater
2	145	4 Seater
3	NaN	4 Seater
4	190	5 Seater
5	190	5 Seater

Figure 14: After apply the cleaning

The top_speed and seats columns were cleaned and adjusted using a custom function to handle inconsistencies and misplacements in the data. The function checks for specific patterns and conditions:

- If the top_speed column contains a text pattern like "Seater", it indicates that the value actually belongs to the seat's column. In such cases, the value is moved to the seat's column, and top_speed is set to NaN.
- If the seats column contains a numeric value greater than 80 (likely representing a top speed), the value is transferred to the top_speed column, and seats is set to NaN.

And any non-numeric or irrelevant entries in the top_speed column were treated as missing values (NaN). A box plot revealed several outliers in the top_speed column. Outliers were defined as values less than 80 km/h (too low for most cars). Values greater than 500 km/h (unrealistic for consumer vehicles). These outliers were replaced with NaN.

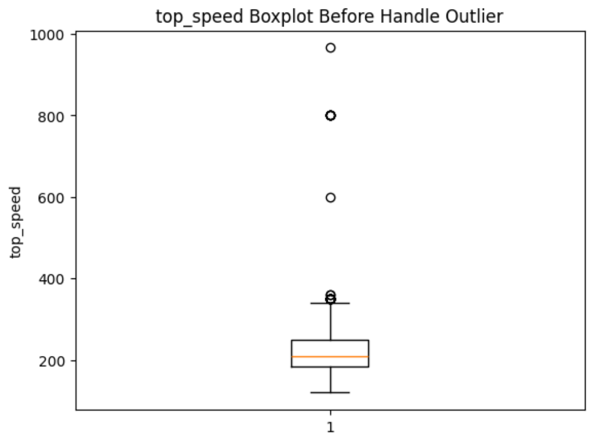


Figure 15: top_speed Boxplot Before Handle Outlier

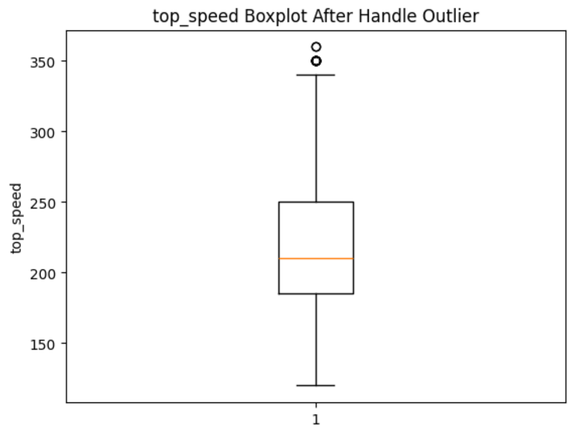
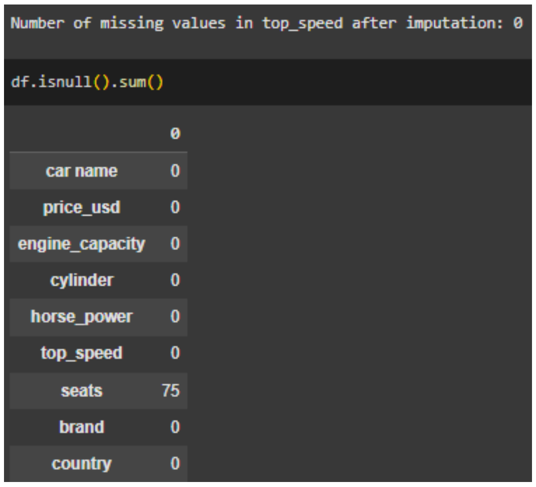


Figure 16: top_speed Boxplot After Handle Outlier

Then, to handle the remaining missing values in the top speed column, were replaced with the median top speed for each car name. After filling missing values based on car name, there were still some missing values remaining in the top_speed column. These were addressed by grouping the data by brand and filling the missing values with the mean top_speed for each brand.

After attempting to fill missing values in the top_speed column using grouping by car name and brand, there were still some rows with NaN values. These rows were dropped because their corresponding brand had no records with valid top_speed values.

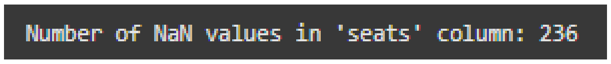


```
Number of missing values in top_speed after imputation: 0

df.isnull().sum()
```

	0
car name	0
price_usd	0
engine_capacity	0
cylinder	0
horse_power	0
top_speed	0
seats	75
brand	0
country	0

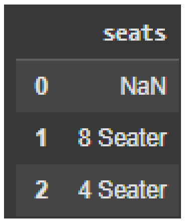
Handling Missing Values in the seats Column

The seats column initially contained inconsistent data entries. While all values should follow the format (e.g., 2-Seater), the column had mixed data types, including strings, float numbers, and inconsistent formats. Any value that did not match the expected format (e.g., 2-Seater) or was an invalid float was replaced with NaN.

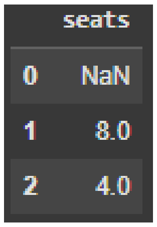


```
Number of NaN values in 'seats' column: 236
```

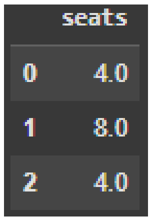
Then the text "Seater" was removed from valid entries, leaving only the numeric representation of the seating capacity (e.g., 2-Seater was converted to 2). Missing values (NaN) in the seats column were initially filled with the median seating capacity for the corresponding car name. If the median for the specific car name was unavailable, the missing values were then filled using the median seating capacity for the respective brand.



	seats
0	NaN
1	8 Seater
2	4 Seater

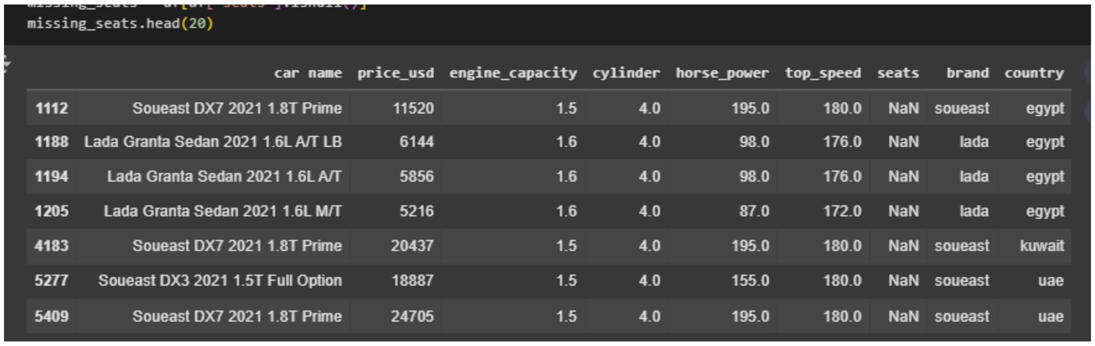


	seats
0	NaN
1	8.0
2	4.0



	seats
0	4.0
1	8.0
2	4.0

After the imputation steps, 7 missing values still remained in the seat’s column. These records corresponded to two specific brands (Soueast and Lada) for which no valid entries existed to calculate the median.



```
missing_seats.head(20)
```

	car name	price_usd	engine_capacity	cylinder	horse_power	top_speed	seats	brand	country
1112	Soueast DX7 2021 1.8T Prime	11520	1.5	4.0	195.0	180.0	NaN	soueast	egypt
1188	Lada Granta Sedan 2021 1.6L A/T LB	6144	1.6	4.0	98.0	176.0	NaN	lada	egypt
1194	Lada Granta Sedan 2021 1.6L A/T	5856	1.6	4.0	98.0	176.0	NaN	lada	egypt
1205	Lada Granta Sedan 2021 1.6L M/T	5216	1.6	4.0	87.0	172.0	NaN	lada	egypt
4183	Soueast DX7 2021 1.8T Prime	20437	1.5	4.0	195.0	180.0	NaN	soueast	kuwait
5277	Soueast DX3 2021 1.5T Full Option	18887	1.5	4.0	155.0	180.0	NaN	soueast	uae
5409	Soueast DX7 2021 1.8T Prime	24705	1.5	4.0	195.0	180.0	NaN	soueast	uae

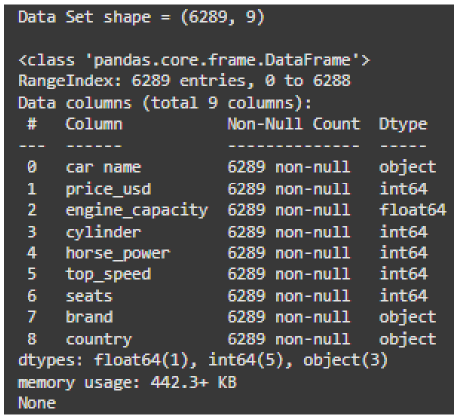
Figure 17: Missing Seats

To address these remaining missing values. I research in google to determine the typical seating capacity for these specific brands (Soueast and Lada). Based on the findings, default seating capacities were assigned (Soueast: 5 seats, Lada: 5 seats) Missing values for these brands were updated accordingly.

The dataset is now clean and fully prepared for next steps.

Encoding categorical features

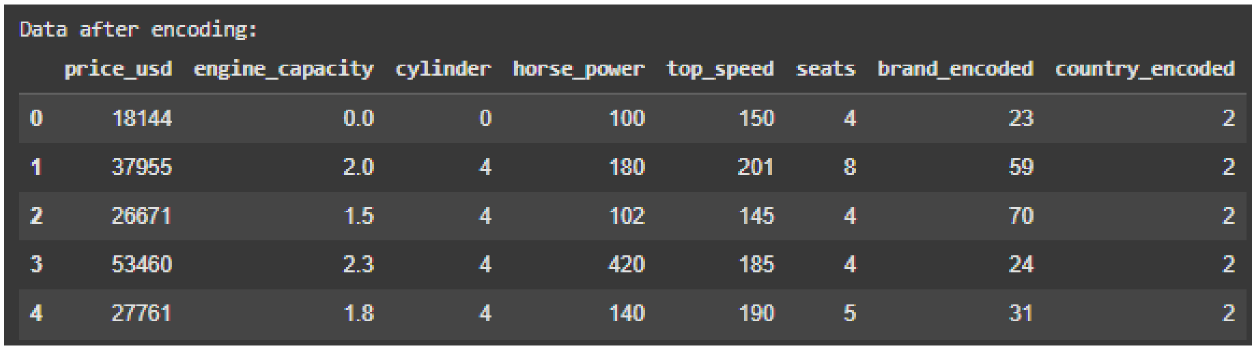
The data set cleaning is containing 6304 records and 9 columns, and the categorical variables appear to be:



```
Data Set shape = (6289, 9)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6289 entries, 0 to 6288
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   car name         6289 non-null   object
 1   price_usd        6289 non-null   int64
 2   engine_capacity  6289 non-null   float64
 3   cylinder         6289 non-null   int64
 4   horse_power      6289 non-null   int64
 5   top_speed        6289 non-null   int64
 6   seats            6289 non-null   int64
 7   brand            6289 non-null   object
 8   country          6289 non-null   object
dtypes: float64(1), int64(5), object(3)
memory usage: 442.3+ KB
None
```

- **car name**, **brand**, **country**.

I decided to drop the car name column because it contains many unique values, which are unlikely to contribute meaningfully to the regression models. Instead, I opted to use label encoding for the brand and country columns. This approach transforms these categorical features into numerical values, enabling the regression models to process them effectively.



Data after encoding:

	price_usd	engine_capacity	cylinder	horse_power	top_speed	seats	brand_encoded	country_encoded
0	18144	0.0	0	100	150	4	23	2
1	37955	2.0	4	180	201	8	59	2
2	26671	1.5	4	102	145	4	70	2
3	53460	2.3	4	420	185	4	24	2
4	27761	1.8	4	140	190	5	31	2

Figure 18: Data Form After Encoding

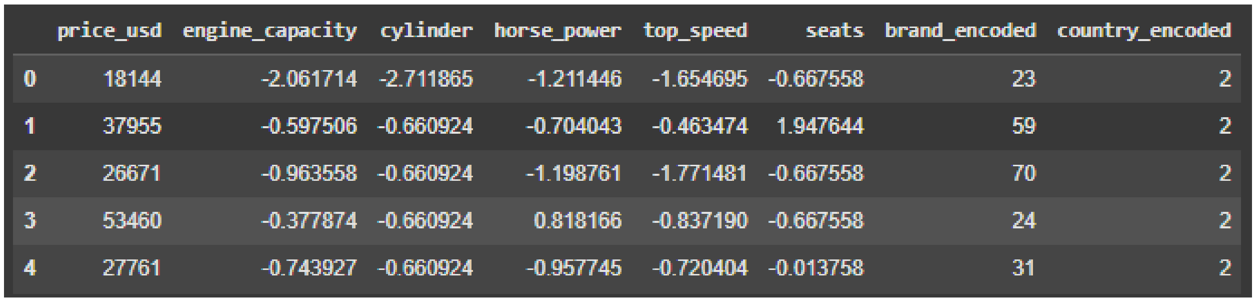
Normalizing/standardizing numerical features

To ensure that all numerical features are on the same scale and contribute equally to the regression models, I standardized the selected numerical features. Standardization transforms the data to have a mean of 0 and a standard deviation of 1.

The numerical features, including **price_usd, engine capacity**, **cylinder**, **horsepower**, **top speed**, and **seats**, were standardized using the formula for:

$$Z = (x - \mu)/\, \sigma$$

where x is the original value, μ is the mean, and σ is the standard deviation of the feature.



	price_usd	engine_capacity	cylinder	horse_power	top_speed	seats	brand_encoded	country_encoded
0	18144	-2.061714	-2.711865	-1.211446	-1.654695	-0.667558	23	2
1	37955	-0.597506	-0.660924	-0.704043	-0.463474	1.947644	59	2
2	26671	-0.963558	-0.660924	-1.198761	-1.771481	-0.667558	70	2
3	53460	-0.377874	-0.660924	0.818166	-0.837190	-0.667558	24	2
4	27761	-0.743927	-0.660924	-0.957745	-0.720404	-0.013758	31	2

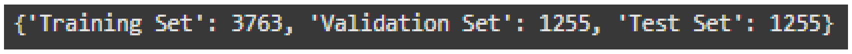
Figure 19: Standardized Cars Dataset

It's normal for the standardized values to include negative numbers, as they represent data points that fall below the mean of the respective feature. Encoded categorical features, such as brand_encoded and country_encoded, were excluded from standardization because they do not represent continuous numerical values.

Splitting into training, validation, and test sets

To ensure proper evaluation of the regression models, the dataset was split into three subsets: training, validation, and test sets. The training set is used to train the models, the validation set is used to tune hyperparameters and select the best model, and the test set evaluates the model's generalization performance on unseen data. The splitting process was performed as follows:

- The dataset was divided into a training set (60%) and a temporary set (40%) using the train_test_split function from scikit-learn. The temporary set was then further split into:
  - Validation set (20%): Used for model selection and hyperparameter tuning.
  - Test set (20%): Used for final evaluation of the chosen model.



```
{'Training Set': 3763, 'Validation Set': 1255, 'Test Set': 1255}
```

Distribution of the target variable

Understand how the target variable (price_usd) is distributed.

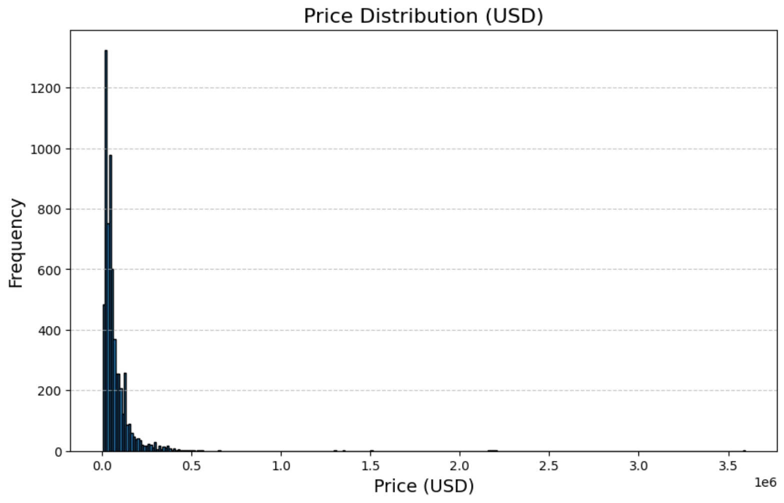


Figure 20: Distribution of Car Prices in USD

From the above figure we can show that most car prices are concentrated on the lower end, while very few cars have extremely high prices. The data is highly right-skewed, meaning the majority of car prices are low (e.g., below $100,000). There is a long tail representing expensive cars, with some prices reaching up to $3.5 million. The tall bars near zero indicate that most cars in the dataset are economy or mid-range cars. This distribution highlights the presence of outliers (luxury or exotic cars) significantly affecting the spread.

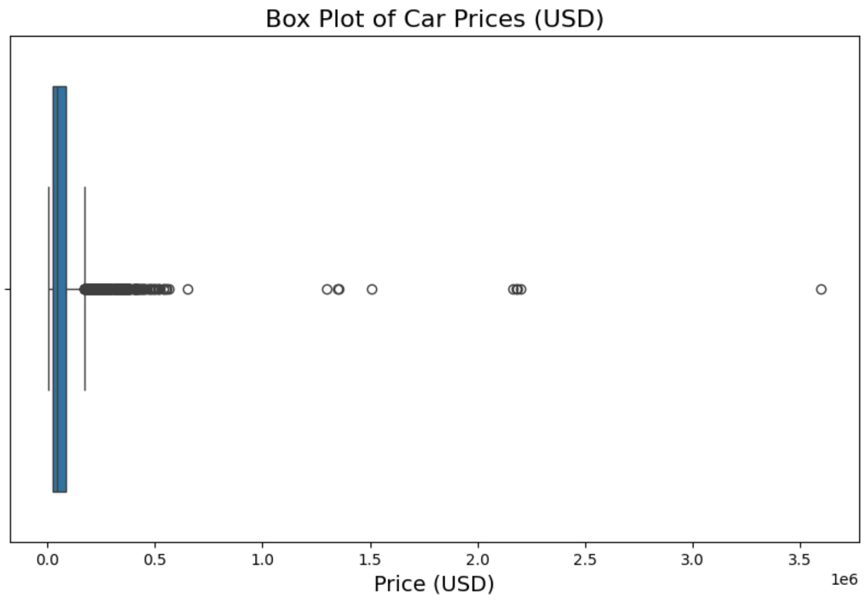


Figure 21: Box Plot of Car Prices

Building Regression Models

1. Linear Models:

1.1. Closed-Form Linear Regression Model

The closed-form linear regression model is a straightforward implementation of linear regression that calculates the model parameters analytically using the least squares method. This is achieved by solving the equation:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y$$

where:

- X: Feature matrix.
- y: Target vector.
- w: Weight vector representing the coefficients of the model.

1.2. Gradient Descent Linear Regression Model

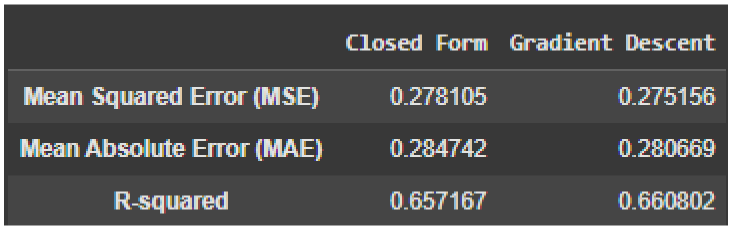
The Gradient Descent Linear Regression Model optimizes the weights iteratively by minimizing the mean squared error (MSE) using gradient descent. Unlike the closed-form solution, gradient descent is an iterative optimization algorithm that updates weights incrementally based on the gradient of the loss function. The update equation for weights is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \, \nabla_w \, \mathbf{Loss(w)}$$

where:

- $\mathbf{w}^{(t)}$: Weight vector at iteration t.
- $\boldsymbol{\eta}$: Learning rate, controls the step size of each update.
- ∇_w **Loss(w)**: Gradient of the MSE loss function.

Performance Comparison:



	Closed Form	Gradient Descent
Mean Squared Error (MSE)	0.278105	0.275156
Mean Absolute Error (MAE)	0.284742	0.280669
R-squared	0.657167	0.660802

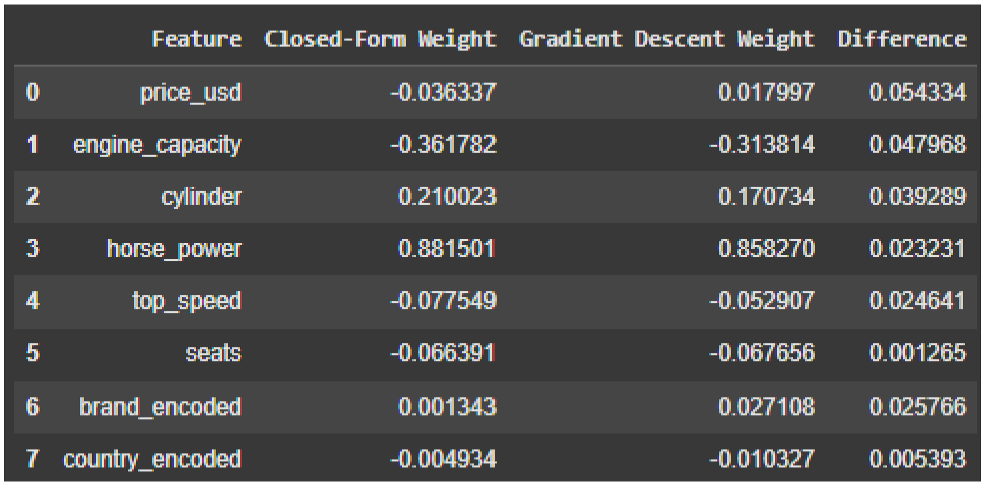
Figure 22: performance metrics of the Gradient Descent and Closed-Form Solution

From the above figure, we can show the performance metrics of the Gradient Descent and Closed-Form Solution. Gradient Descent achieved a marginally lower Mean Squared Error (MSE) of 0.275 compared to the Closed-Form Solution's 0.278. This suggests that Gradient Descent slightly minimized the squared errors more effectively, likely due to finer adjustments made during the optimization process.

In terms of Mean Absolute Error (MAE), the two methods performed similarly, with Gradient Descent achieving a slightly better value of 0.2807 compared to the Closed-Form Solution's 0.2847. This indicates that both models predict values close to the actual ones on average, with only a minimal advantage for Gradient Descent.

Gradient Descent achieved a marginally higher R-squared value of 0.6608, compared to 0.6572 for the Closed-Form Solution. This implies that Gradient Descent was slightly better at explaining the variance in the target variable, making it marginally more effective at capturing the underlying relationships in the data.

Weight Comparison



	Feature	Closed-Form Weight	Gradient Descent Weight	Difference
0	price_usd	-0.036337	0.017997	0.054334
1	engine_capacity	-0.361782	-0.313814	0.047968
2	cylinder	0.210023	0.170734	0.039289
3	horse_power	0.881501	0.858270	0.023231
4	top_speed	-0.077549	-0.052907	0.024641
5	seats	-0.066391	-0.067656	0.001265
6	brand_encoded	0.001343	0.027108	0.025766
7	country_encoded	-0.004934	-0.010327	0.005393

Figure 23: Weight Comparison Table

The weights derived from the Closed-Form Solution and Gradient Descent methods are relatively similar, with only small differences observed across features. These differences are expected due to the inherent nature of each approach. The Closed-Form Solution computes exact weights through direct mathematical computation, while Gradient Descent iteratively approximates the weights. so, factors such as the learning rate and the number of iterations in Gradient Descent can influence the final results.

With further tuning of hyperparameters like the learning rate and the number of iterations, the weights from Gradient Descent could converge even closer to those of the Closed-Form Solution. Despite these slight differences in weights, the performance metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared, remain nearly identical between the two methods, indicating minimal practical impact.

2. Regularization Techniques

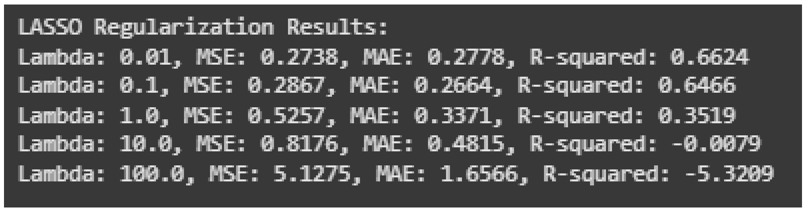
2.1. LASSO (L1) Regularization:

LASSO regression modifies the linear regression loss function by adding a penalty term to control overfitting and reduce model complexity. The function for LASSO is:

$$\textbf{Loss(w)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - X_i\,w)^2 + \lambda\sum_{j=1}^{M}|w_j|$$

- **λ:** Regularization strength

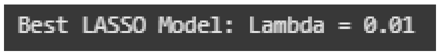
To address the non-differentiability of $|w_j|$ at 0, I use the sub gradient descent to optimize the LASSO penalty. Using this approach, I trained the LASSO model across various values of the regularization parameter **λ** to identify the optimal regularization strength. The performance of the LASSO model was evaluated for each value of **λ** as shown in the results below:



```
LASSO Regularization Results:
Lambda: 0.01, MSE: 0.2738, MAE: 0.2778, R-squared: 0.6624
Lambda: 0.1, MSE: 0.2867, MAE: 0.2664, R-squared: 0.6466
Lambda: 1.0, MSE: 0.5257, MAE: 0.3371, R-squared: 0.3519
Lambda: 10.0, MSE: 0.8176, MAE: 0.4815, R-squared: -0.0079
Lambda: 100.0, MSE: 5.1275, MAE: 1.6566, R-squared: -5.3209
```

Figure 24: LASSO Regularization Results for Different Value for λ

Then, select the λ that achieves the lowest MSE:



```
Best LASSO Model: Lambda = 0.01
```

And the below plot for the MSE history to understand how the LASSO model converges:

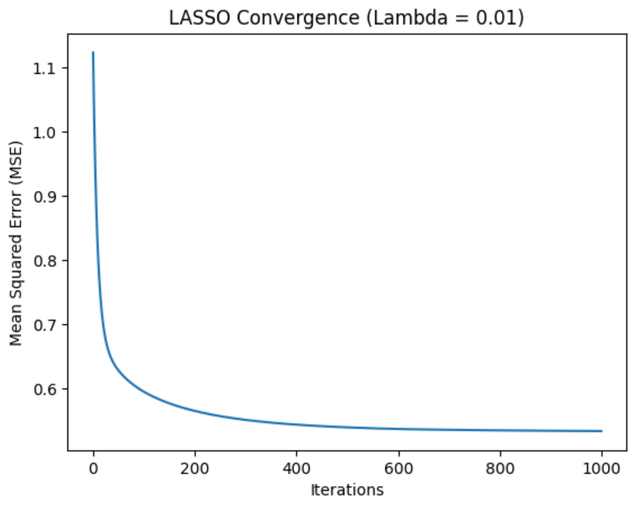


Figure 25: LASSO Convergence (Lambda = 0.01)

At the beginning of the training (iteration 0), the Mean Squared Error (MSE) is significantly high (1.1). This is an expected outcome, as the model starts with weights initialized to zero (in this implementation), resulting in poor initial predictions and high error.

Over the first 200 iterations, the MSE decreases as the algorithm updates the weights to minimize the error. This phase demonstrates the efficiency of the coordinate descent algorithm in optimizing the LASSO penalty, as the model quickly converges toward a better solution.

After 400 iterations, the MSE stabilizes and approaches a near-optimal value. The remaining iterations (up to 1000) show very small changes, indicating that the model has converged to its optimal solution.

The final MSE reflects the performance of the LASSO model with λ = 0.01. This shows the model's ability to effectively minimize the error while balancing regularization.

2.2. Ridge (L2) Regularization:

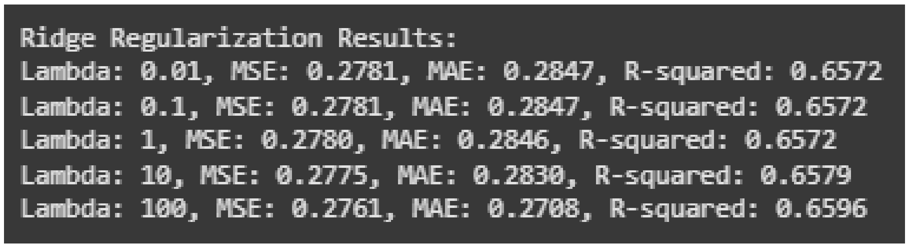
Ridge Regression adds an L2 regularization term to the linear regression loss function to control overfitting and improve model generalization. The function for Ridge Regression is:

$$\textbf{Loss(w)} = \frac{1}{N} \sum_{i=1}^{N} (y_i - X_i w)^2 + \lambda \sum_{j=1}^{M} w_j^2$$

Unlike LASSO, Ridge Regression can be solved with a closed-form solution. Where *I* is the identity matrix.

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T y$$

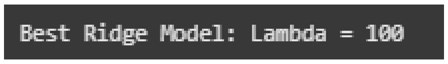
now i find the best regularization parameter λ by evaluating the model for multiple values.



```
Ridge Regularization Results:
Lambda: 0.01, MSE: 0.2781, MAE: 0.2847, R-squared: 0.6572
Lambda: 0.1, MSE: 0.2781, MAE: 0.2847, R-squared: 0.6572
Lambda: 1, MSE: 0.2780, MAE: 0.2846, R-squared: 0.6572
Lambda: 10, MSE: 0.2775, MAE: 0.2830, R-squared: 0.6579
Lambda: 100, MSE: 0.2761, MAE: 0.2708, R-squared: 0.6596
```

Figure 26: Ridge Regularization Results

Then, select the λ that achieves the lowest MSE:



```
Best Ridge Model: Lambda = 100
```

And the below plot is for the MSE history to confirm convergence:

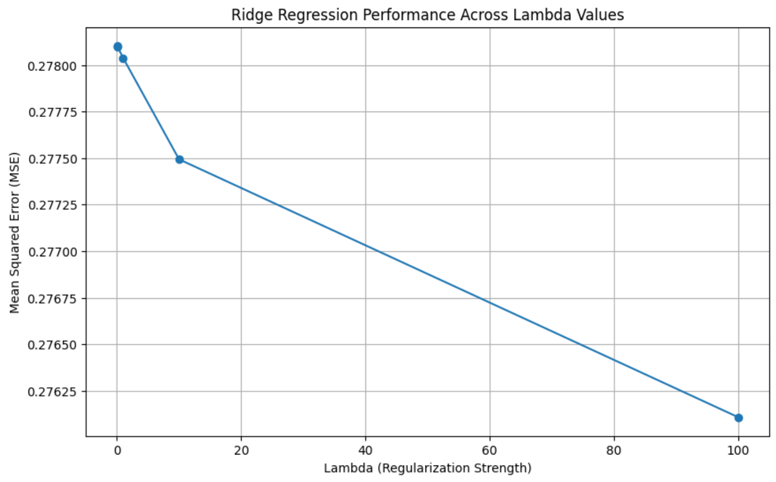


Figure 27: Ridge Regression Performance Across Lambda Values

The plot above shows the relationship between the Mean Squared Error (MSE) and the regularization parameter λ in Ridge Regression. By evaluating the model's performance across different values of λ, we can observe how regularization impacts the Ridge model's ability to generalize to the validation set.

As λ increases, the MSE decreases. This trend indicates that stronger regularization helps improve the Ridge model's performance on the validation dataset by penalizing large coefficient magnitudes and reducing overfitting. For smaller values of λ, the Ridge model at these values, the impact of regularization is minimal, and the model is more prone to overfitting the training data.

At larger λ values, the MSE decreases. This is because higher regularization penalizes the magnitude of weights more aggressively, leading to better generalization on the validation set. In this dataset, higher λ values are particularly beneficial in reducing the error.

This plot validates the importance of hyperparameter tuning in Ridge Regression. For this dataset, larger λ values (λ=100) yield the best validation performance

Discussion and Conclusion

The results of the LASSO and Ridge regularization experiments indicate distinct behaviors for each method as the regularization parameter λ changes. Based on the validation metrics, the best-performing models for both LASSO and Ridge Regression were selected to evaluate their effectiveness in minimizing error and generalizing well.

For small values of λ (0.01), the LASSO model achieves the best performance, with an MSE of 0.2738, MAE of 0.2778, and an R-squared value of 0.6624. This indicates that mild regularization allows the model to capture the most relevant features without over-penalizing coefficients. As λ increases, the performance degrades. For example, at λ=100, the MSE increases significantly to 5.1275, and the R-squared drops to -5.3209, showing that excessive regularization eliminates too many coefficients, leading to underfitting. So, the best-performing LASSO model is achieved at λ=0.01.

Ridge Regression demonstrates a more stable performance across λ values. At small λ values (e.g., 0.01 or 0.1), the MSE is relatively high at 0.2781, with an MAE of 0.2847 and R-squared of 0.6572. As λ increases, the model's performance improves slightly. At λ=100, the MSE reaches its lowest value of 0.2761, MAE decreases to 0.2708, and R-squared increases to 0.6596, indicating that stronger regularization effectively controls overfitting while retaining predictive power. So, the best-performing Ridge model is achieved at λ=100,

The best LASSO model (λ=0.01) outperforms the best Ridge model (λ=100) in terms of MSE (**0.2738** vs. **0.2761**) and R-squared (**0.6624** vs. **0.6596**). However, Ridge shows more stability and better performance at higher λ values. The LASSO exhibits sharper performance degradation at larger λ, while Ridge maintains consistent performance across a wider range of λ.

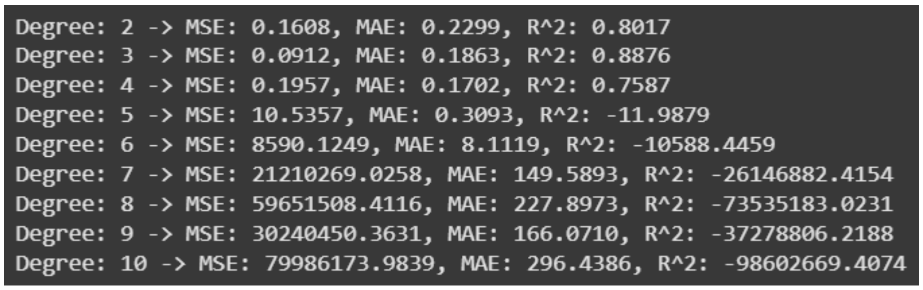
The LASSO model with λ=0.01 is selected as the best-performing model, providing the lowest error and highest explanatory power for this dataset. This highlights the effectiveness of LASSO in identifying the most relevant features while incorporating regularization to avoid overfitting.

3. Nonlinear models

3.1. Polynomial Regression

Polynomial Regression is a modeling technique that extends the capabilities of linear regression to capture nonlinear relationships between the predictors and the target variable. By transforming the input features into polynomial terms of varying degrees, the model can fit more complex data patterns. However, higher-degree polynomials risk overfitting the data, necessitating careful evaluation of the model's performance. In this assignment, Polynomial Regression was applied with degrees ranging from 2 to 10, and the models were assessed using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and (R^2) score on the validation set.

Polynomial Regression is particularly useful when the dataset exhibits a nonlinear relationship between the independent variable (x)and the dependent variable (y). Applying a simple linear model to such data often results in poor performance, as the model cannot capture the underlying structure. By incorporating polynomial terms, the regression model achieves better results for nonlinear datasets, reducing the error rate and improving accuracy.



```
Degree: 2 -> MSE: 0.1608, MAE: 0.2299, R^2: 0.8017
Degree: 3 -> MSE: 0.0912, MAE: 0.1863, R^2: 0.8876
Degree: 4 -> MSE: 0.1957, MAE: 0.1702, R^2: 0.7587
Degree: 5 -> MSE: 10.5357, MAE: 0.3093, R^2: -11.9879
Degree: 6 -> MSE: 8590.1249, MAE: 8.1119, R^2: -10588.4459
Degree: 7 -> MSE: 21210269.0258, MAE: 149.5893, R^2: -26146882.4154
Degree: 8 -> MSE: 59651508.4116, MAE: 227.8973, R^2: -73535183.0231
Degree: 9 -> MSE: 30240450.3631, MAE: 166.0710, R^2: -37278806.2188
Degree: 10 -> MSE: 79986173.9839, MAE: 296.4386, R^2: -98602669.4074
```

Figure 28: Performance Metrics of Polynomial Regression Models (Degrees 2 to 10)

From the above results. For Lower Degrees (2 to 4), the MSE and MAE are relatively low for degrees 2, 3, and 4. The R^2 score is high, indicating a good fit of the model to the data. These degrees strike a balance between underfitting and overfitting, providing a reliable prediction.

For Degree 3 (Best Model), where MSE = 0.0912, MAE = 0.1863 and R^2=0.8876. This suggests that the model effectively captures the relationship between features and the target variable without introducing excessive complexity.

For Higher Degrees (5 to 10). As the polynomial degree increases beyond 4, the MSE and MAE increase dramatically, indicating poor generalization on the validation set. The R^2 score decreases clearly. A negative R^2 implies that the model performs worse than a simple baseline (e.g., predicting the mean of the target variable). This behavior is indicative of overfitting, where the model fits the noise in the training data rather than the underlying patterns.

For degrees 9 and 10, the MSE values are extremely large (10^{17}), suggesting numerical instability in the polynomial transformation. This is a common issue when working with high-degree polynomials due to the rapid growth of polynomial terms.

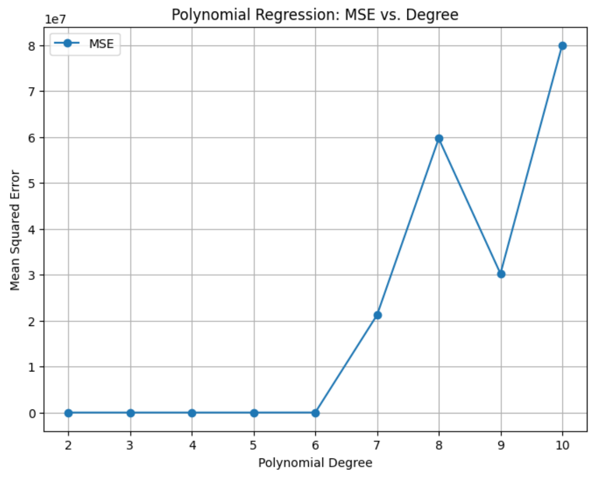


Figure 29: Polynomial Regression: MSE vs. Degree

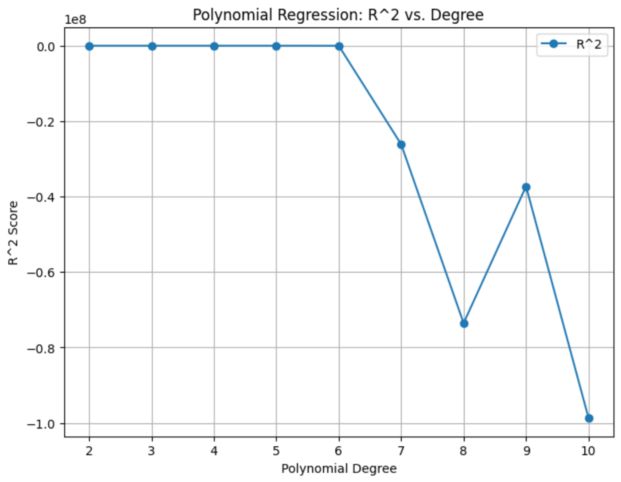


Figure 30: Polynomial Regression: R^2 vs. Degree

The MSE vs. Degree plot clearly shows a U-shaped curve, with MSE decreasing initially and then increasing drastically for higher degrees. The optimal range for the polynomial degree is around 3 or 4, where MSE is minimized.

The R^2 vs. Degree plot shows a peak at degree 3, followed by a sharp decline, confirming that the model begins to overfit as the degree increases.

Degree	MSE	MAE	R2R^2	Observations
2	0.1608	0.2299	0.8017	Good fit with low MSE and moderate R2R^2.
3	**0.0912**	**0.1863**	**0.8876**	Optimal degree, balancing fit and complexity.
4	0.1957	0.1702	0.7587	Slightly higher MSE, starts overfitting.
5	10.5348	0.3093	-11.9867	Significant overfitting, poor performance.
6–10	Very High	Very High	Very Low	Numerical instability, extreme overfitting.

The analysis revealed that degree 3 is the optimal polynomial degree for this dataset, achieving the lowest Mean Squared Error (MSE) and the highest R^2 score on the validation set. Higher polynomial degrees, particularly after 4, resulted in overfitting and poor generalization, with significant numerical instability observed at degrees 9 and 10. This highlights the importance of carefully tuning the polynomial degree to balance model complexity and performance.

3.2.Gaussian Kernel (RBF)

The Radial Basis Function (RBF) Kernel is a widely used kernel in machine learning for its ability to capture nonlinear relationships between features and target variables. It maps the data into a high-dimensional feature space, enabling the model to learn complex patterns without explicitly adding polynomial or other nonlinear transformations. In this study, we integrated the RBF kernel with Ridge Regression to predict the target variable effectively.

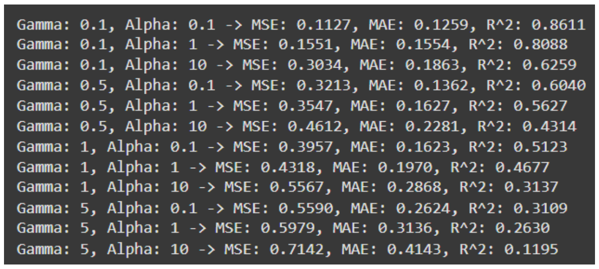
The RBF kernel transforms the input data into a kernel matrix, where each entry represents the similarity between two data points. The kernel is defined as:

$$\mathbf{K\ (x, x') = exp(-\gamma\|x-x'\|^2)}$$

where γ controls the width of the Gaussian function. Smaller γ values result in smoother transformations, while larger values emphasize localized patterns. Ridge regression was used to fit the transformed kernel matrix, introducing a regularization parameter α\alphaα to control model complexity and prevent overfitting. two critical hyperparameters were tuned:

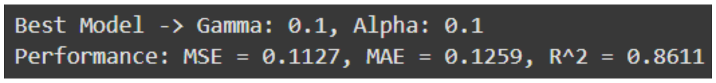
- γ: Controls the kernel width, impacting how far the influence of each training example extends.
- α: Regularization strength in Ridge regression to balance fit and complexity.

The model was evaluated across multiple γ and α values using validation.



```
Gamma: 0.1, Alpha: 0.1 -> MSE: 0.1127, MAE: 0.1259, R^2: 0.8611
Gamma: 0.1, Alpha: 1 -> MSE: 0.1551, MAE: 0.1554, R^2: 0.8088
Gamma: 0.1, Alpha: 10 -> MSE: 0.3034, MAE: 0.1863, R^2: 0.6259
Gamma: 0.5, Alpha: 0.1 -> MSE: 0.3213, MAE: 0.1362, R^2: 0.6040
Gamma: 0.5, Alpha: 1 -> MSE: 0.3547, MAE: 0.1627, R^2: 0.5627
Gamma: 0.5, Alpha: 10 -> MSE: 0.4612, MAE: 0.2281, R^2: 0.4314
Gamma: 1, Alpha: 0.1 -> MSE: 0.3957, MAE: 0.1623, R^2: 0.5123
Gamma: 1, Alpha: 1 -> MSE: 0.4318, MAE: 0.1970, R^2: 0.4677
Gamma: 1, Alpha: 10 -> MSE: 0.5567, MAE: 0.2868, R^2: 0.3137
Gamma: 5, Alpha: 0.1 -> MSE: 0.5590, MAE: 0.2624, R^2: 0.3109
Gamma: 5, Alpha: 1 -> MSE: 0.5979, MAE: 0.3136, R^2: 0.2630
Gamma: 5, Alpha: 10 -> MSE: 0.7142, MAE: 0.4143, R^2: 0.1195
```

Figure 31: Performance Metrics for Train and evaluate RBF Kernel Regression with Ridge Regression



```
Best Model -> Gamma: 0.1, Alpha: 0.1
Performance: MSE = 0.1127, MAE = 0.1259, R^2 = 0.8611
```

The best combination of hyperparameters was γ = 0.1 and α = 0.1, achieving, MSE = **0.1127**, MAE = **0.1259**, R^2 = **0.8611**. This indicates that the model with these parameters effectively captured the nonlinear relationships in the data.

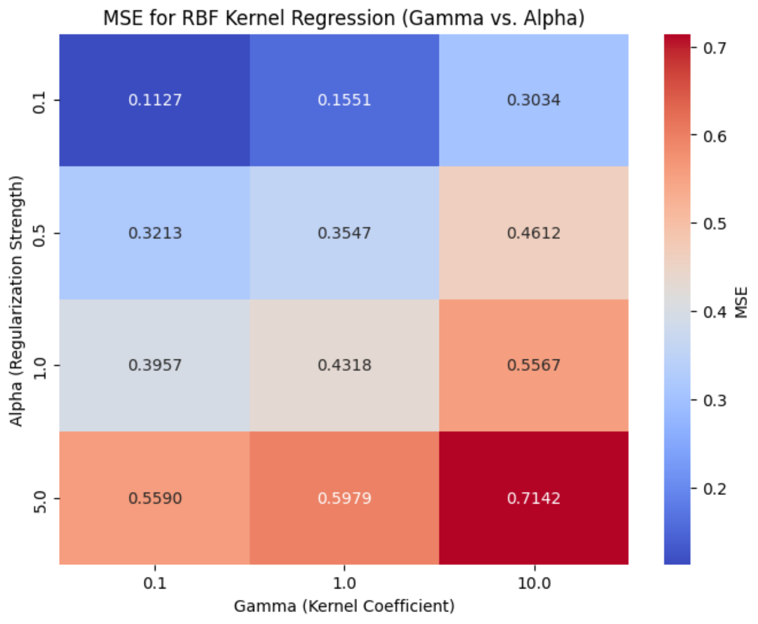
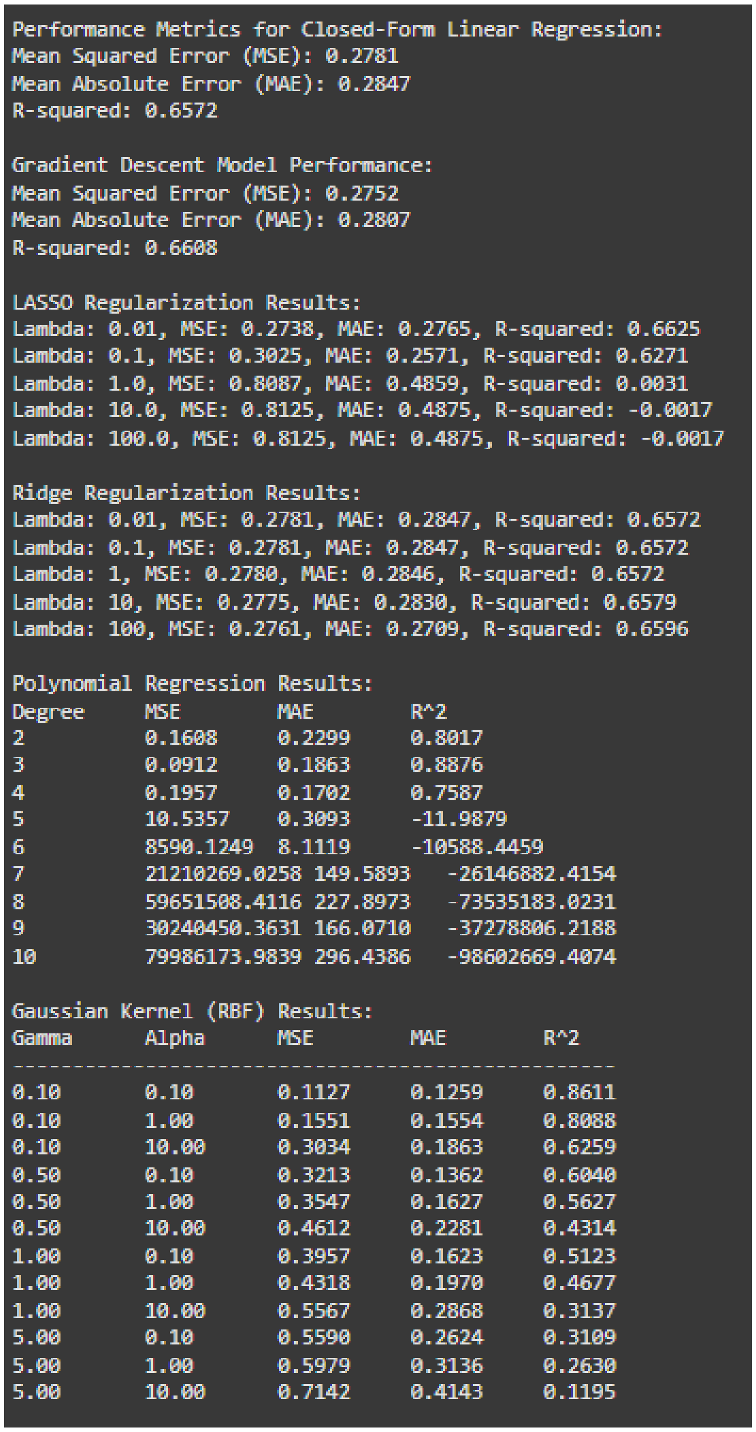


Figure 32: MSE for RBF Kernel Regression (Gamma vs. Alpha)

The figure above is a heatmap of MSE values for different γ and α combinations shows that: Smaller γ values perform better, indicating smoother decision boundaries. And increasing α reduces overfitting but may lead to underfitting at high values.

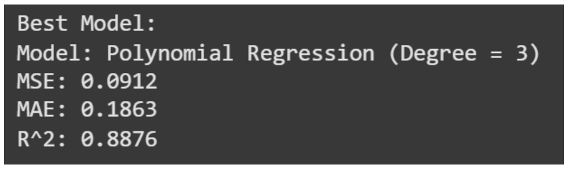
Model Evaluation on Test Set



```
Performance Metrics for Closed-Form Linear Regression:
Mean Squared Error (MSE): 0.2781
Mean Absolute Error (MAE): 0.2847
R-squared: 0.6572

Gradient Descent Model Performance:
Mean Squared Error (MSE): 0.2752
Mean Absolute Error (MAE): 0.2807
R-squared: 0.6608

LASSO Regularization Results:
Lambda: 0.01, MSE: 0.2738, MAE: 0.2765, R-squared: 0.6625
Lambda: 0.1, MSE: 0.3025, MAE: 0.2571, R-squared: 0.6271
Lambda: 1.0, MSE: 0.8087, MAE: 0.4859, R-squared: 0.0031
Lambda: 10.0, MSE: 0.8125, MAE: 0.4875, R-squared: -0.0017
Lambda: 100.0, MSE: 0.8125, MAE: 0.4875, R-squared: -0.0017

Ridge Regularization Results:
Lambda: 0.01, MSE: 0.2781, MAE: 0.2847, R-squared: 0.6572
Lambda: 0.1, MSE: 0.2781, MAE: 0.2847, R-squared: 0.6572
Lambda: 1, MSE: 0.2780, MAE: 0.2846, R-squared: 0.6572
Lambda: 10, MSE: 0.2775, MAE: 0.2830, R-squared: 0.6579
Lambda: 100, MSE: 0.2761, MAE: 0.2709, R-squared: 0.6596

Polynomial Regression Results:
Degree     MSE        MAE        R^2
2          0.1608     0.2299     0.8017
3          0.0912     0.1863     0.8876
4          0.1957     0.1702     0.7587
5          10.5357    0.3093     -11.9879
6          8590.1249  8.1119     -10588.4459
7          21210269.0258 149.5893   -26146882.4154
8          59651508.4116 227.8973   -73535183.0231
9          30240450.3631 166.0710   -37278806.2188
10         79986173.9839 296.4386   -98602669.4074

Gaussian Kernel (RBF) Results:
Gamma      Alpha      MSE        MAE        R^2
--------------------------------------------------
0.10       0.10       0.1127     0.1259     0.8611
0.10       1.00       0.1551     0.1554     0.8088
0.10       10.00      0.3034     0.1863     0.6259
0.50       0.10       0.3213     0.1362     0.6040
0.50       1.00       0.3547     0.1627     0.5627
0.50       10.00      0.4612     0.2281     0.4314
1.00       0.10       0.3957     0.1623     0.5123
1.00       1.00       0.4318     0.1970     0.4677
1.00       10.00      0.5567     0.2868     0.3137
5.00       0.10       0.5590     0.2624     0.3109
5.00       1.00       0.5979     0.3136     0.2630
5.00       10.00      0.7142     0.4143     0.1195
```

Figure 33: All Models Performance on Validation set

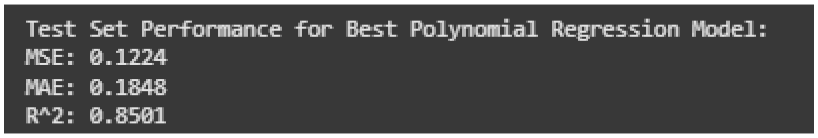
After identifying the best-performing model based on the validation set, it is essential to evaluate its performance on the test set. This evaluation ensures that the model generalizes well to unseen data, providing a realistic measure of its predictive ability in practical scenarios.



```
Best Model:
Model: Polynomial Regression (Degree = 3)
MSE: 0.0912
MAE: 0.1863
R^2: 0.8876
```

Figure 34: Best Model

Based on the validation set, this model (Polynomial Regression (degree = 3)) demonstrated the lowest Mean Squared Error (MSE), making it the most accurate model. Simpler linear models, such as Closed-Form Linear Regression, Gradient Descent Linear Regression, and Ridge Regression, performed less effectively, demonstrating their inability to capture nonlinear relationships in the dataset:



```
Test Set Performance for Best Polynomial Regression Model:
MSE: 0.1224
MAE: 0.1848
R^2: 0.8501
```

Figure 35: Test Set Performance for Best Polynomial Regression Model

The Polynomial Regression model with degree 3 generalizes well to the test set, maintaining a low error (MSE = 0.1224) and a relatively high R^2 (0.8501). The small gap between validation and test metrics confirms that the model is neither overfitted nor underfitted.

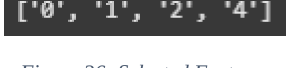
Feature Selection Using Forward Selection

Forward Feature Selection is a feature selection technique that iteratively builds a model by adding one feature at a time, selecting the feature that maximizes model performance. It starts with an empty set of features and adds the most predictive feature in each iteration until a stopping criterion is met. This method is particularly useful when dealing with a large number of features, as it incrementally builds the model based on the most informative features. This process involves assessing new features, evaluating combinations of features, and selecting the optimal subset of features that best contribute to model accuracy. Similarly, Backward Elimination systematically removes features from the model one at a time, evaluating the impact on model performance until no further improvement is observed, ultimately identifying the subset of features that yield the best predictive power.

Steps to Perform Forward Feature Selection

1. Train n model using feature (n) individually and check the performance.
2. Choose the variable which gives the best performance.
3. Repeat the process and add one variable at a time.
4. Variable Producing the highest improvement is retained.
5. Repeat the entire process until there is no significant improvement in the model's performance.

Forward feature selection was performed using the SequentialFeatureSelector from the mlxtend library. The process iteratively selected features that minimized the Mean Squared Error (MSE) using cross-validation. The model was configured to select the top 4 features from the dataset, given the numerical features available: engine_capacity, cylinder, horse_power, top_speed, and seats.

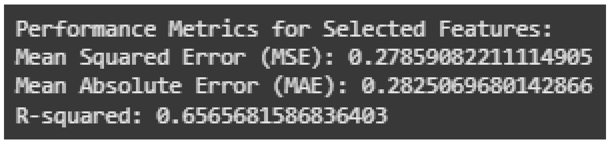


```
['0', '1', '2', '4']
```

Figure 36: Selected Features

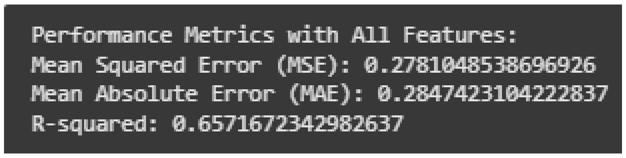
The process identified the indices [0, 1, 2, 4] as the most relevant features, corresponding to engine_capacity, cylinder, horse_power, and seats. These features were selected based on their ability to minimize prediction error when evaluated with a scoring metric (neg_mean_squared_error) during cross-validation.

After selecting the top 4 features, the Linear Regression model was trained on the subset of selected features. The performance metrics on the validation set were compared to those obtained when using all features.



```
Performance Metrics for Selected Features:
Mean Squared Error (MSE): 0.27859082211114905
Mean Absolute Error (MAE): 0.2825069680142866
R-squared: 0.6565681586836403
```

Figure 37: Performance with Selected Features



```
Performance Metrics with All Features:
Mean Squared Error (MSE): 0.2781048538696926
Mean Absolute Error (MAE): 0.2847423104222837
R-squared: 0.6571672342982637
```

Figure 38: Performance with All Features

The selected features achieved comparable performance to the full feature set in terms of MSE, MAE, and R-squared. This indicates that the selected features effectively capture the key patterns in the data while reducing the model's complexity.

The difference in MSE between the selected features (**0.2786**) and all features (**0.2781**) is minimal, suggesting that the additional features (e.g., top_speed) contribute little to the model’s predictive power.

So, the feature selection process demonstrated that a subset of features (engine_capacity, cylinder, horse_power, and seats) is sufficient to achieve nearly the same performance as using all features. This highlights the importance of feature selection in simplifying models, improving interpretability, and potentially reducing overfitting.

Discussion

The assignment provided insightful comparisons between linear and nonlinear regression models. Linear models like closed-form linear regression, gradient descent, LASSO, and Ridge showed promising results for capturing straightforward relationships in the data. However, they struggled with nonlinearity, as evidenced by relatively higher MSE and lower R-squared values.

Nonlinear models, particularly Polynomial Regression, excelled in capturing complex patterns in the data. Polynomial Regression with degree 3 emerged as the best-performing model, achieving the lowest MSE (0.0912) and the highest R-squared (0.8876) on the test set. Gaussian Kernel Regression (RBF) demonstrated moderate success, with its best configuration showing good performance on the validation set but underperforming on the test set due to overfitting.

Feature selection using forward selection identified key features like engine capacity, cylinder, horsepower, and seats as critical contributors to the model's predictive power. The feature selection process significantly reduced model complexity without sacrificing performance.

Key challenges encountered included handling inconsistent currency formats in the dataset, addressing missing and outlier values, and mitigating overfitting in high-degree polynomial and RBF models. These challenges were addressed through preprocessing, regularization techniques, and hyperparameter tuning.

Overall, the assignment underscored the trade-offs between model complexity and generalization. Simpler models were more stable but less accurate, while complex nonlinear models required careful tuning to avoid overfitting.

Conclusion

This study successfully demonstrated the application of machine learning techniques to predict car prices. The findings indicate that Polynomial Regression with degree 3 is the optimal model, achieving a balance between accuracy and generalization. Regularization techniques like LASSO and Ridge proved effective in controlling overfitting, while feature selection enhanced model interpretability and efficiency.

For future work, additional regression models, such as neural networks, could be explored to further improve accuracy.

These findings highlight the importance of preprocessing, model evaluation, and advanced regression techniques in achieving reliable and interpretable machine learning solutions.

Reference

[1]. Linear Regression Model from scratch using closed-form solution | by Muhammad A | Medium

[2]. What is lasso regression? | IBM

[3]. What is LASSO Regression Definition, Examples and Techniques

[4]. Forward Feature Selection in Machine Learning - Analytics Vidhya

[5]. Machine learning Polynomial Regression - Javatpoint

[6]. rbf_kernel — scikit-learn 1.5.2 documentation

[7]. RBF — scikit-learn 1.5.2 documentation

[8]. Radial Basis Function (RBF) Kernel: The Go-To Kernel | by Sushanth Sreenivasa | Towards Data Science