Electrical and Computer Engineering Department

Machine Learning and Data Science

ENCS5341

Assignment #2: Regression Analysis and Model Selection

Prepared By:

Name	ID
Rahaf Naser	1201319
Rania Rimawi	1201179

Instructor's Name: Ismail Khater

Section: 2

November 28, 2024

Table of contents

Part1.Description of the dataset, preprocessing steps

The first 5 rows of the dataset before preprocessing:

	car name	price	engine_capacity	cylinder	horse_power	top_speed	seats	brand	country
0	Fiat 500e 2021 La Prima	TBD	0.0	N/A, Electric	Single	Automatic	150	fiat	ksa
1	Peugeot Traveller 2021 L3 VIP	SAR 140,575	2.0	4	180	8 Seater	8.8	peugeot	ksa
2	Suzuki Jimny 2021 1.5L Automatic	SAR 98,785	1.5	4	102	145	4 Seater	suzuki	ksa
3	Ford Bronco 2021 2.3T Big Bend	SAR 198,000	2.3	4	420	4 Seater	7.5	ford	ksa
4	Honda HR-V 2021 1.8 i-VTEC LX	Orangeburst Metallic	1.8	4	140	190	5 Seater	honda	ksa

Information of dataset before preprocessing:

```
<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
```

```
df.describe()
```

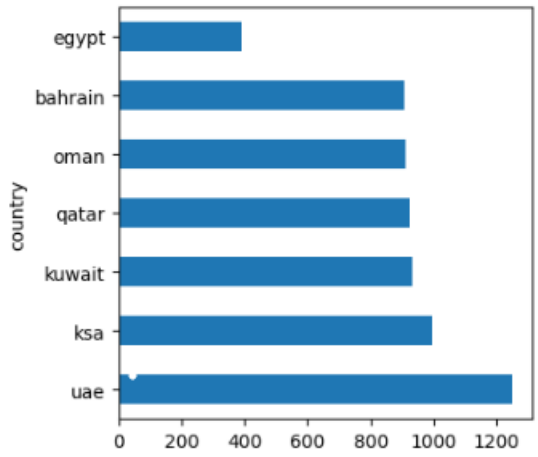
	car name	price	engine_capacity	cylinder	horse_power	top_speed	seats	brand	country
count	6308	6308	6308	5684	6308	6308	6308	6308	6308
unique	2546	3395	129	10	330	169	82	82	7
top	Mercedes-Benz C-Class 2022 C 300	TBD	2.0	4	150	250	5 Seater	mercedes-benz	uae
freq	10	437	1241	2856	162	1100	3471	560	1248

Data for each column before preprocessing:

```
Name: car name dtype: object

car name
Mercedes-Benz C-Class 2022 C 300                        10
Fiat 500e 2021 La Prima                                  7
Porsche 911 2021 Carrera S Cabriolet                     7
Porsche 911 2021 Carrera 4S                              7
Kia Rio Sedan 2021 1.6L MPI                              7
                                                        ..
Jaguar F-Type Convertible 2021 2.0T R-Dynamic (300 PS)   1
Hyundai Kona 2021 1.6L T-GDI GLS Premium (AWD)           1
Chevrolet Trax 2021 1.8L Premier AWD                     1
Jaguar XF 2021 3.0 V6 SC R Sport                         1
Lamborghini Aventador Ultimae 2022 LP 780-4              1
Name: count, Length: 2546, dtype: int64
```

```
Name: brand dtype: object

brand
mercedes-benz    560
audi             398
bmw              394
toyota           378
ford             323
                ...
tata               2
soueast            2
byd                2
dfm                1
bugatti            1
Name: count, Length: 82, dtype: int64


Name: country dtype: object

country
uae       1248
ksa        996
kuwait     932
qatar      925
oman       910
bahrain    906
egypt      391
Name: count, dtype: int64


Name: price dtype: object

price
TBD               437
Following         238
DISCONTINUED      140
Follow             27
Grigio Maratea     23
                 ...
BHD 23,900          1
BHD 24,300          1
BHD 24,100          1
BHD 24,700          1
AED 1,650,000       1
Name: count, Length: 3395, dtype: int64


Name: engine_capacity dtype: object

engine_capacity
2.0     1241
3.0      703
3.5      359
1.5      347
4.0      340
        ...
3342       1
2476       1
4400       1
3470       1
1595       1
Name: count, Length: 129, dtype: int64
```

```
Name: cylinder dtype: object

cylinder
4                2856
6                1480
8                 924
3                 139
12                136
N/A, Electric     107
10                 21
5                  17
Drive Type          3
16                  1
Name: count, dtype: int64


Name: horse_power dtype: object

horse_power
150    162
355    116
400    111
184    109
300    103
      ...
87       1
126      1
394      1
236      1
720      1
Name: count, Length: 330, dtype: int64


Name: top_speed dtype: object

top_speed
250    1100
180     410
200     392
170     332
190     294
       ...
600       1
130       1
966       1
262       1
324       1
Name: count, Length: 169, dtype: int64


Name: seats dtype: object

seats
5 Seater    3471
4 Seater     847
7 Seater     532
2 Seater     428
8 Seater     211
            ...
24.1           1
12.3           1
230            1
220            1
2.8            1
Name: count, Length: 82, dtype: int64
```



Data Cleaning:

For each column in the dataset this are some reflection before start:

- **car name** and **brand**: no more action.
- **price**: create a custom function in order to extract price and currency. Where is it possible create price in dollars
- **engine_capacity**, **cylinder**, **horse_power**, **top_speed**: simple conversion to float and set a limit
- **country**: simple re-mapping
- **seats**: ignore it
- We preprocess the dataset to ensure data quality and standardization for further analysis. Specifically, we adjust the `price` column by extracting currency codes and values, converting them into Dollars using predefined exchange rates. To facilitate this, we use a function to transform the data and create new columns (`price_country`, `price_currency`, `price_dollar`) that capture these adjustments.
- We clean and standardize numerical columns such as `engine_capacity`, `cylinder`, `horse_power`, and `top_speed`. For each, we convert values to numeric, filter them against predefined realistic limits, and replace outliers with '0'. Additionally, we identify electric cars by checking specific patterns in the `cylinder` and `engine_capacity` columns.
- We standardize country names by mapping abbreviations to full forms (e.g., 'uae' to 'United Arab Emirates'). Finally, we remove unnecessary columns, such as intermediary numeric conversions and `seats`, to simplify the dataset. These steps ensure the data is clean, consistent, and ready for analysis.

	car name	price_country	price_currency	price_dollar	engine_capacity_l	cylinder_nr	is_eletric	horse_power_cv	top_speed_kmh	brand	country
2	Suzuki Jimny 2021 1.5L Automatic	98785.0	SAR	26671.95	1.5	4.0	False	102.0	145.0	suzuki	Saudi Arabia
5	Honda HR-V 2021 1.8 i-VTEC EX	95335.0	SAR	25740.45	1.8	4.0	False	140.0	190.0	honda	Saudi Arabia
6	Peugeot Expert 2021 Van L3 A/T	82845.0	SAR	22368.15	2.0	4.0	False	120.0	170.0	peugeot	Saudi Arabia
7	Peugeot Expert 2021 Van L3 M/T	76545.0	SAR	20667.15	2.0	4.0	False	120.0	170.0	peugeot	Saudi Arabia
8	Renault Koleos 2021 2.5L LE (4WD)	116900.0	SAR	31563.0	2.5	4.0	False	170.0	199.0	renault	Saudi Arabia

The dataset after the above processes

Shape of new dataset:

```
(3904, 11)
```

Columns of dataset

```
Index(['car name', 'price_country', 'price_currency', 'price_dollar',
       'engine_capacity_l', 'cylinder_nr', 'is_eletric', 'horse_power_cv',
       'top_speed_kmh', 'brand', 'country'],
      dtype='object')
```

We preprocess and prepare the dataset for modeling by encoding, normalizing, and splitting it into training, validation, and test sets.

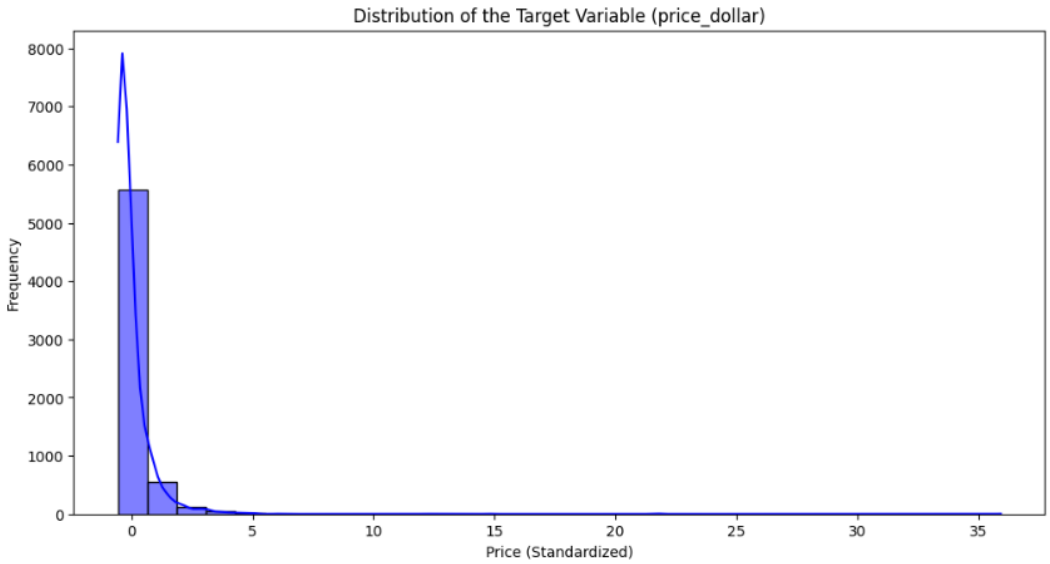
We begin by loading the dataset and encoding categorical features into numerical values using label encoding. This step converts text-based categories into integer representations to make the data compatible with machine learning algorithms. Next, we normalize numerical features using a StandardScaler to ensure they have a mean of 0 and a standard deviation of 1. This scaling process helps models converge faster and reduces sensitivity to feature magnitudes.

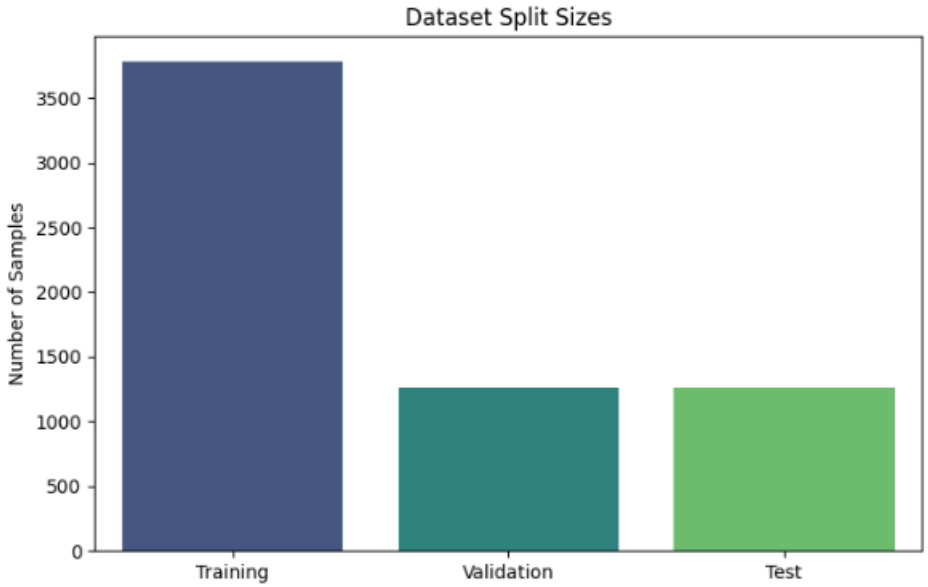
We define the target variable as price_dollar and split the dataset into three subsets: training (60%), validation (20%), and test (20%). This split is performed to ensure we can train the model effectively, validate its performance on unseen data, and evaluate its generalizability.

We visualize the processed data by plotting the distribution of the target variable (price_dollar), which helps us understand its range and central tendencies. Additionally, we create a bar plot to display the sizes of the training, validation, and test splits, ensuring the division aligns with our desired proportions. These steps set the foundation for robust model training and evaluation.

The result:

```
Training set: (3784, 10) (3784,)
Validation set: (1262, 10) (1262,)
Test set: (1262, 10) (1262,)
```





Part2. Building Regression Models

2.1. Linear Models: Linear Regression, LASSO, and Ridge Regression

- We first identify and drop columns with all Nan values from the training and validation sets. This ensures we're not working with any unnecessary columns that could introduce errors.
- For the remaining missing values, we apply imputation. We use the median strategy for numerical columns (to avoid distorting the data with outliers) and the most frequent strategy for categorical columns (to fill missing values with the most common category).

We train three different models:

- **Linear Regression**: A basic regression model with no regularization. It minimizes the residual sum of squares between the observed and predicted values.
- **LASSO Regression (L1 Regularization)**: This model applies L1 regularization, which can help with feature selection by shrinking some coefficients to zero, potentially improving model interpretability and preventing overfitting.
- **Ridge Regression (L2 Regularization)**: Unlike LASSO, Ridge applies L2 regularization, which penalizes large coefficients but does not shrink them to zero. It helps manage multicollinearity and overfitting.

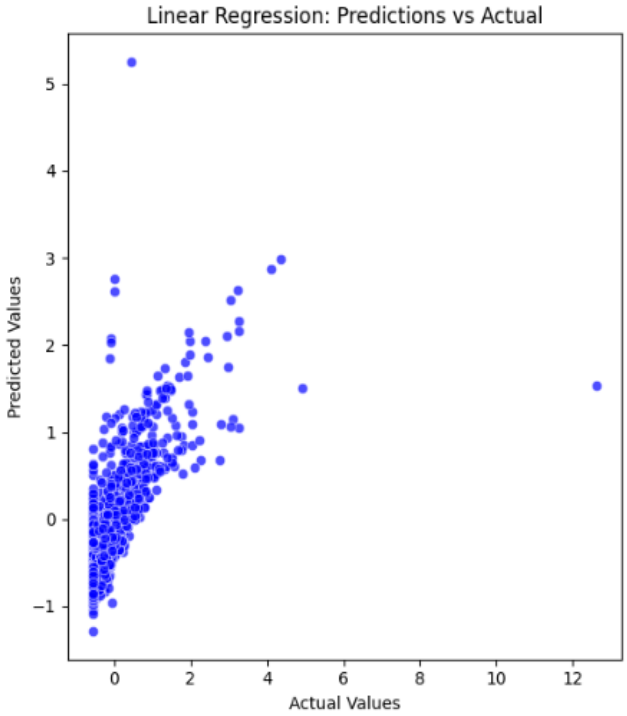
For each model, we calculate the **Mean Squared Error (MSE)**, which measures the average squared difference between the actual and predicted values. A lower MSE indicates better model performance.

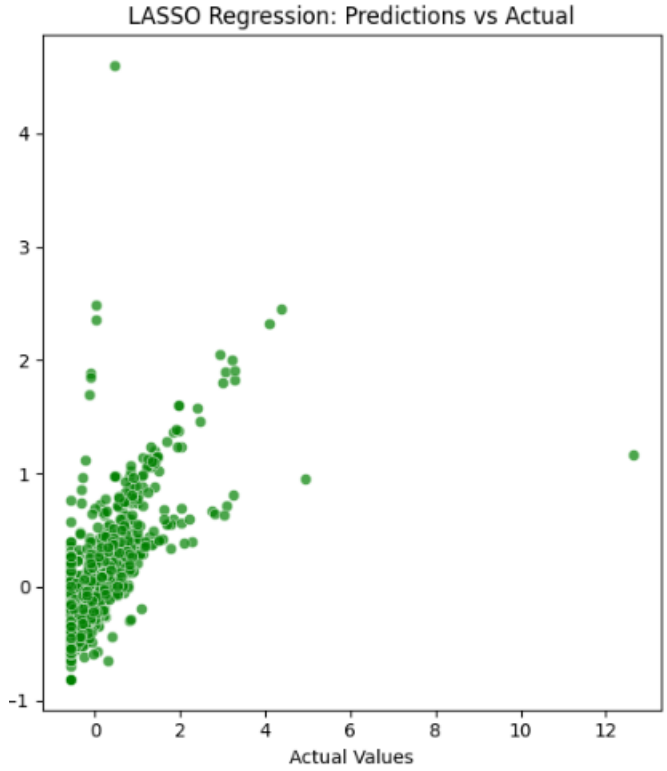
The result:

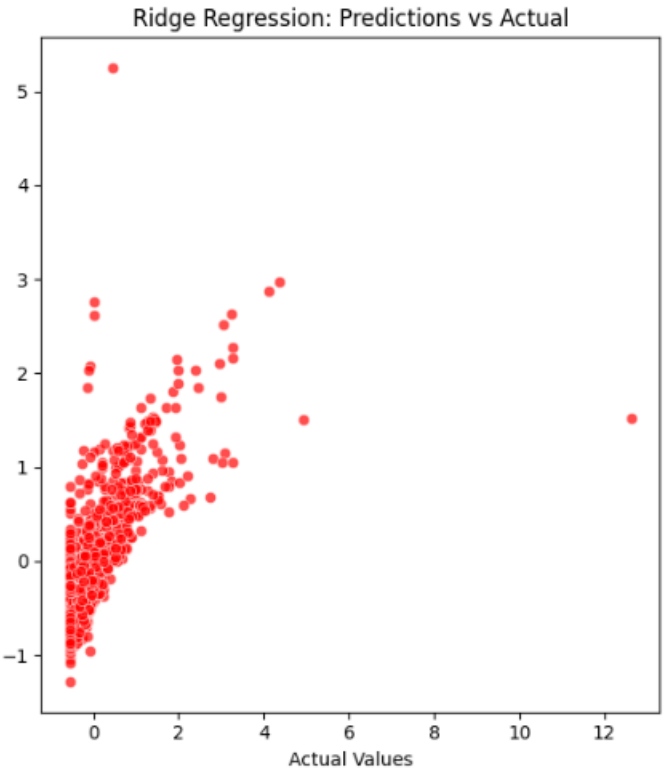
```
Linear Regression MSE: 0.29257306021185714
LASSO Regression MSE: 0.30349443108747737
Ridge Regression MSE: 0.2924468787772562
```

These results indicate that:

- **Linear Regression** and **Ridge Regression** perform almost identically, with very close MSE values (0.292573 and 0.292447, respectively), suggesting they are similarly effective at predicting the target variable.
- **LASSO Regression** has a slightly higher MSE (0.303494), which could imply that the regularization applied in LASSO may be too strong for this dataset, leading to a marginally worse fit.

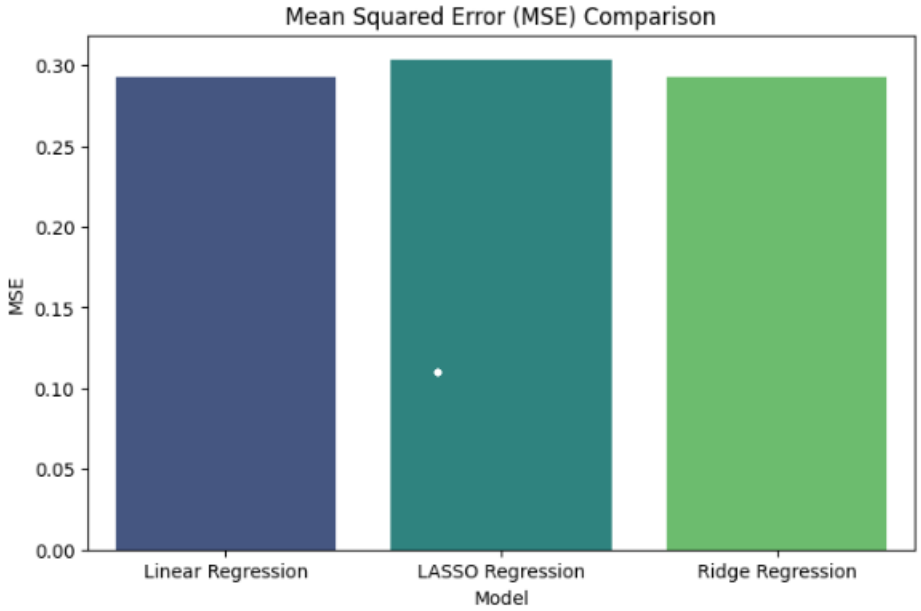

Linear Regression: Predictions vs Actual
Predicted Values
Actual Values
5
4
3
2
1
0
−1
0
2
4
6
8
10
12


LASSO Regression: Predictions vs Actual
Actual Values
4
3
2
1
0
-1
0
2
4
6
8
10
12



We visualize the **Predictions vs Actual** values for each model using scatter plots. These plots show how well each model's predictions align with the actual values. Ideally, the points should lie close to the diagonal line, indicating accurate predictions.

We also create a **bar chart comparing MSE values** across the three models to give a clear, quantitative comparison of their performance.



The results suggest that Ridge and Linear Regression are more effective for this dataset compared to LASSO, as reflected in the MSE values.

2.2. Closed-Form Solution for Linear Regression (Without Libraries)

We implement **Closed-Form Linear Regression** to directly compute the optimal model parameters (theta) using the **Normal Equation**.

We first select only the numeric columns from the training and validation datasets, ensuring we use features that are compatible with the regression model. This step excludes any non-numeric data, which would interfere with the mathematical calculations.

We add a bias term (intercept) to the feature matrices by inserting a column of ones at the beginning of the feature arrays. This allows the model to learn an intercept, in addition to the weights for the features, ensuring a more flexible model.

We ensure the feature matrices are in float64 format to avoid any issues with matrix operations, as the closed-form solution involves inversion and multiplication of matrices.

Using the **Normal Equation**, we calculate the best-fit parameters (theta) for the linear regression model. The Normal Equation is given by:

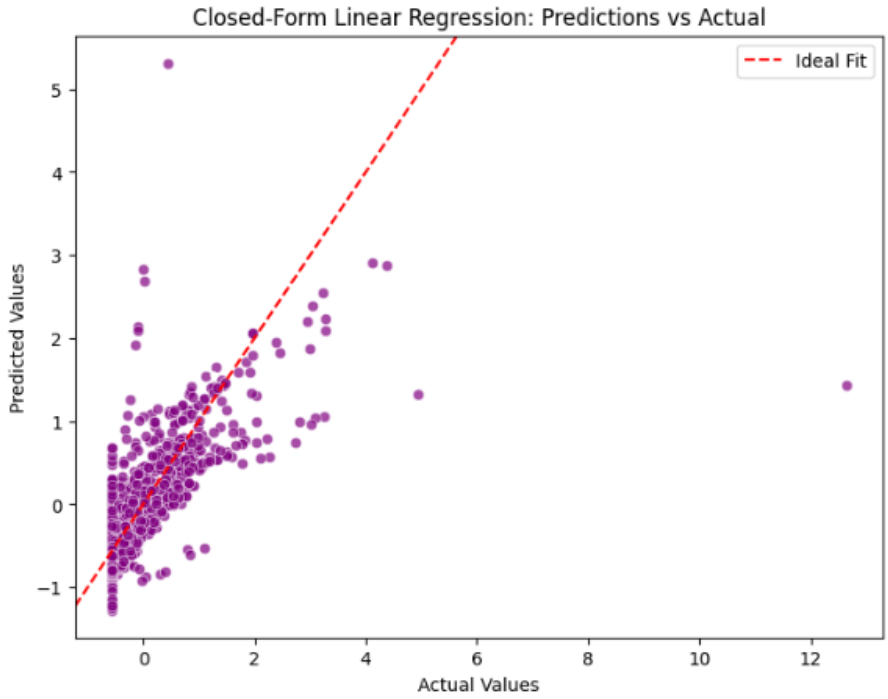
$$\theta = (X^T X)^{-1} X^T y$$

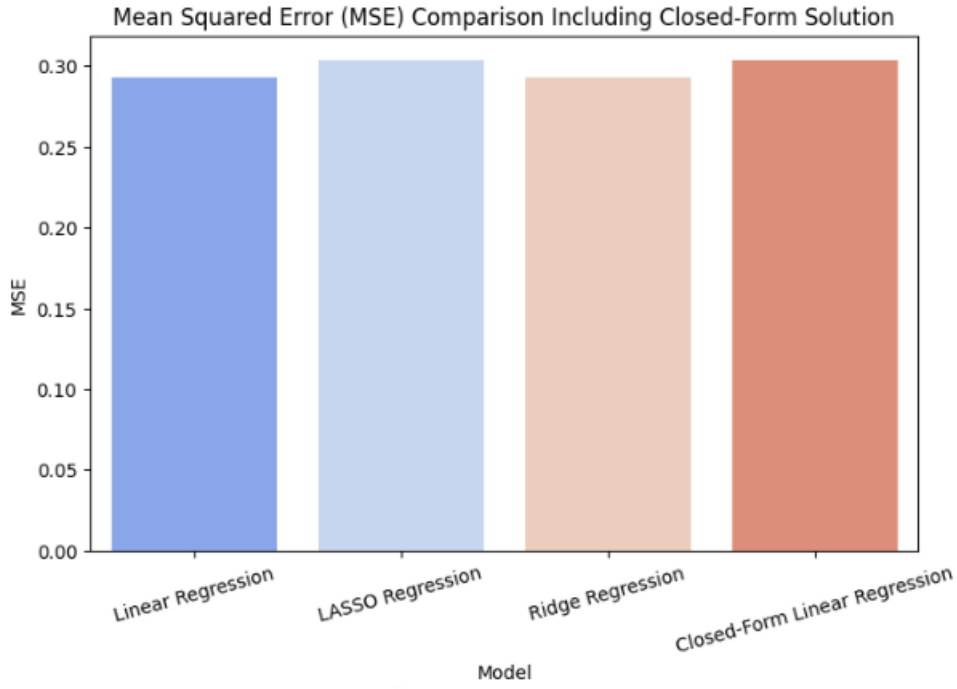
This equation directly computes the optimal parameters without the need for iterative optimization.

We then use the calculated theta values to predict the target variable (y_pred_closed_form) for the validation set. The **Mean Squared Error (MSE)** is calculated to evaluate the performance of the model by comparing the predicted values with the actual values in the validation set.

The result:

```
Closed-Form Linear Regression MSE: 0.30344901778271427
```





The **MSE of Closed-Form Linear Regression** (0.3034) is compared against the MSEs of other models (like **Linear Regression**, **LASSO**, and **Ridge**) to see how well it performs relative to those models.

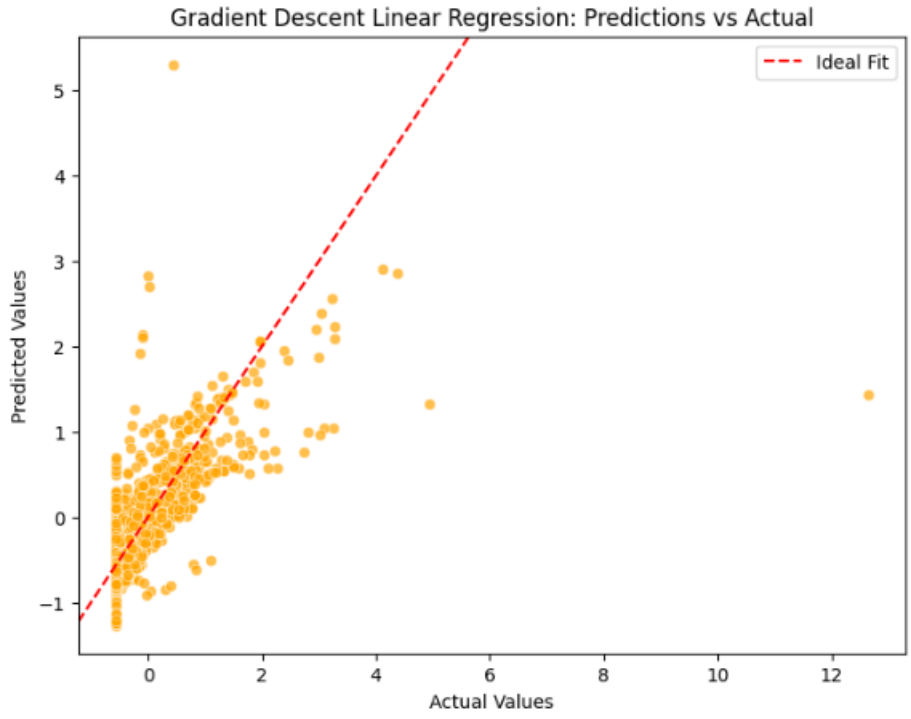
- **Linear Regression MSE**: 0.292573
- **LASSO Regression MSE**: 0.303494

- **Ridge Regression MSE**: 0.292447
- From these results, we can see that the **Closed-Form Linear Regression** performs similarly to the **Linear Regression** and **Ridge Regression** models, with an MSE close to 0.2924. It performs slightly worse than **Linear Regression** but slightly better than **LASSO**, which had the highest MSE.

2.3. Linear Regression using Gradient Descent

The result:

```
Gradient Descent Linear Regression MSE: 0.3028725089140265
```



The **Gradient Descent Linear Regression** model aims to find the optimal parameters for linear regression by iteratively minimizing the mean squared error (MSE) between the predicted and actual values. Using a learning rate of 0.01 and 1000 iterations, the model converges to the optimal parameters and produces predictions. The result of **MSE: 0.3029** indicates that, on average, the squared error between predicted and actual values is around 0.303. While this suggests a reasonable fit, the performance should be compared to other models to determine if Gradient Descent Linear Regression is the most effective choice.

Gradient Descent (0.3029) performs similarly to the **Closed-Form Linear Regression** model (0.3034), both yielding relatively low MSE values, suggesting good performance with slightly different methods.

2.4. Nonlinear Models: Polynomial Regression and Radial Basis Function (RBF)

2.4.1.Polynomial Regression (Degrees 2 to 10)

The result:

```
Polynomial Regression (degree 2) MSE: 0.13556384133595525
Polynomial Regression (degree 3) MSE: 1.1397092185762787
Polynomial Regression (degree 4) MSE: 0.06325923190221139
Polynomial Regression (degree 5) MSE: 1398.1668309563938
Polynomial Regression (degree 6) MSE: 33.50760848529681
Polynomial Regression (degree 7) MSE: 36014.14854533237
Polynomial Regression (degree 8) MSE: 4399640.3840159
Polynomial Regression (degree 9) MSE: 10800539.887108551
```

We implementing **Polynomial Regression** for different polynomial degrees (from 2 to 10) and calculating the **Mean Squared Error (MSE)** for each degree.

For each degree (from 2 to 10), the code creates polynomial features using PolynomialFeatures(degree=degree), transforms the training and validation datasets, and then fits a **Linear Regression** model to the transformed features.

After training the model, predictions are made on the validation set, and the **MSE** is calculated to evaluate the model's performance. MSE is used to determine how close the predictions are to the actual values — a lower MSE indicates better model performance.

Results and Model Performance:

- **Degree 2**: MSE = **0.1356** — The model performs well with a small MSE, indicating that a quadratic polynomial is a good fit for the data.
- **Degree 3**: MSE = **1.1397** — The model's performance worsens, as the cubic polynomial introduces more complexity than needed, leading to higher error.
- **Degree 4**: MSE = **0.0633** — The quartic polynomial improves the fit significantly, resulting in a low MSE and better performance.
- **Degree 5**: MSE = **1398.1668** — A dramatic rise in MSE, indicating overfitting. The polynomial complexity is too high, and the model starts fitting noise.
- **Degree 6**: MSE = **33.5076** — The model continues to overfit with an even higher MSE.
- **Degree 7**: MSE = **36014.1485** — MSE increases drastically, further indicating overfitting.
- **Degree 8**: MSE = **4399640.3840** — The model shows extreme overfitting, with MSE growing uncontrollably.
- **Degree 9**: MSE = **10800539.8871** — The model is severely overfitting, with MSE increasing significantly.

- **Lower-degree polynomials (degree 2 and 4)**: These models provide a good balance between fitting the data and avoiding overfitting. The MSEs are relatively low, indicating that they generalize well.
- **Higher-degree polynomials (degree 5 and above)**: These models show a sharp increase in MSE, indicating overfitting. Higher-degree polynomials introduce unnecessary complexity, causing the model to fit noise and fail to generalize to new data.
- The **best polynomial degree** is **2**, with an MSE of 0.1356, as it provides a good fit without overfitting. Degree 4 also performs well with a low MSE of 0.0633.
- As the polynomial degree increases beyond 4, the models show severe overfitting, with MSE values skyrocketing. This demonstrates that choosing the right polynomial degree is crucial, as too high a degree can make the model overly complex and cause poor generalization.

2.4.2. Radial Basis Function (RBF) / Gaussian Kernel

The result :

```
RBF Kernel Regression MSE: 9120996244.59574
```

Radial Basis Function (RBF) transformation is applied to the data. The RBF kernel is a popular transformation technique in machine learning that maps the input data into a higher-dimensional space. It uses the Gaussian (bell curve) function to compute the similarity between data points.

In the function rbf_kernel(), the centers are selected as the first 100 data points from X_train. The RBF kernel transformation is applied to both the training and validation sets to create new features that represent the similarity of data points to the centers.

MSE: 9120996244.59574: The model's performance is measured using **Mean Squared Error (MSE)**, which quantifies how far the predicted values are from the actual values. A **lower MSE** indicates a better fit of the model.

The reported MSE value of **9120996244.59574** is **quite large**, suggesting that the model performs poorly in terms of predicting the target variable (price_dollar).

The high MSE could be due to several reasons:

-Overfitting: The RBF kernel, especially with the chosen centers, may have overfitted the training data, capturing noise and resulting in poor generalization on the validation set.

-Gamma Parameter: The choice of gamma=0.1 might not be optimal. A very small or large gamma can lead to poor model performance. Tuning this hyperparameter can help improve the model's predictions.

Part3: Model Selection Using Validation Set

The result:

```
Training set: (3784, 10) (3784,)
Validation set: (1262, 10) (1262,)
Test set: (1262, 10) (1262,)
Linear Regression - MSE: 0.2926, MAE: 0.3057, R-squared: 0.4532
LASSO Regression - MSE: 0.5361, MAE: 0.4554, R-squared: -0.0021
Ridge Regression - MSE: 0.2924, MAE: 0.3053, R-squared: 0.4534

Best model based on MSE: Ridge Regression
Best model based on MAE: Ridge Regression
Best model based on R-squared: Ridge Regression
```

The validation set is used to evaluate the performance of different models during training, **before testing** the final model on the test data. This helps in tuning hyperparameters and selecting the best-performing model without overfitting the model to the test data.

The validation set is used to assess how well each model is likely to perform on unseen data.

The model with the best performance (lowest MSE, lowest MAE, and highest R-squared) on the validation set is chosen as the best model.

In this case, Ridge Regression was chosen as the best model based on its superior performance on the validation set, making it the model selected for further evaluation on the test data.

Part4: Feature Selection with Forward Selection

The result:

```
Added feature: horse_power_cv, Validation MSE: 0.3298
Added feature: price_currency, Validation MSE: 0.3029
Added feature: is_eletric, Validation MSE: 0.2941
Added feature: top_speed_kmh, Validation MSE: 0.2879
Added feature: car name, Validation MSE: 0.2869
Selected features: ['horse_power_cv', 'price_currency', 'is_eletric', 'top_speed_kmh', 'car name']
```

This method implements **Forward Selection** for feature selection using a linear regression model and a validation set to guide the process. Let's go step by step through how the method works and how to interpret the results.

Steps:

1. Initialization:
2. Maximum Features:
3. Forward Selection Loop:
4. Stopping Condition:
5. Returning the selected features: After the loop, the selected features are returned.

Final Selected Features:

- The selected features are:
    - horse_power_cv
    - price_currency
    - is_eletric
    - top_speed_kmh
    - car name

These are the features that have been identified as most useful for the model, as they lead to the lowest MSE on the validation set.

Results Interpretation:

- Added feature: horse_power_cv, Validation MSE: 0.3298:
    - The feature horse_power_cv was added to the model first, and it resulted in an MSE of 0.3298, which is the lowest MSE at this step.
- Added feature: price_currency, Validation MSE: 0.3029:
    - The feature price_currency was added next. The MSE improved to 0.3029, showing that adding this feature reduced the error.
- Added feature: is_eletric, Validation MSE: 0.2941:
    - Adding the feature is_eletric further improved the MSE to 0.2941.
- Added feature: top_speed_kmh, Validation MSE: 0.2879:
    - The feature top_speed_kmh resulted in an even lower MSE of 0.2879.
- Added feature: car name, Validation MSE: 0.2869:
    - Finally, the feature car name was added, leading to a further slight improvement in the MSE to 0.2869.

Forward selection helps reduce the complexity of the model by selecting only the most important features, which can improve generalization and performance, and prevent overfitting. The resulting model is simpler, more interpretable, and often faster to train.

Part5: Regularization Techniques Use LASSO and Ridge regularization

We are comparing LASSO regression and Ridge regression, two machine learning methods that add regularization (a penalty) to improve model performance. The goal is to find the best penalty value () and see which method works better.

Based on result :

```
Optimal LASSO λ: 0.00015998587196060574
LASSO CV - MSE: 0.2924, MAE: 0.3052, R-squared: 0.4535
Optimal Ridge λ: 10.0
Ridge CV - MSE: 0.2919, MAE: 0.3025, R-squared: 0.4545

Model Comparison:
LASSO - Best λ: 0.00015998587196060574, MSE: 0.2924, MAE: 0.3052, R2: 0.4535
Ridge - Best λ: 10.0, MSE: 0.2919, MAE: 0.3025, R2: 0.4545
```

The figure shows the result comparing LASSO AND Ridge

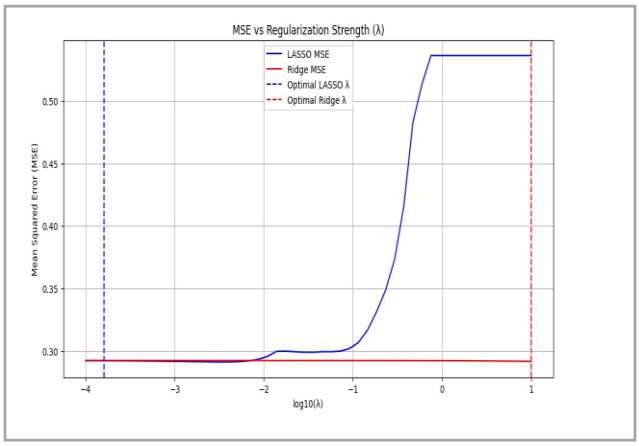
Best Regularization strength (λ)	
LASSO **0.00016**	Ridge 10

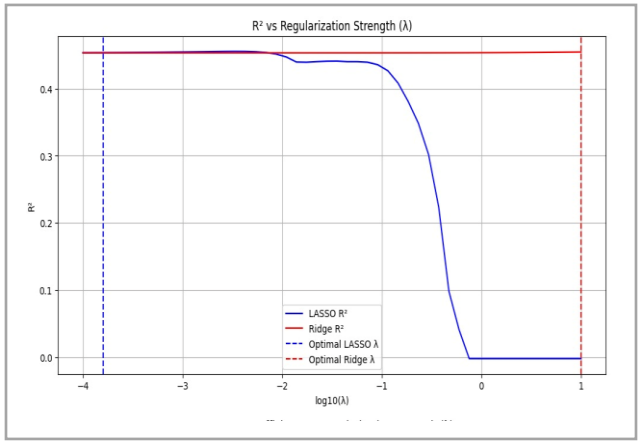
Table1: Best Regularization strength

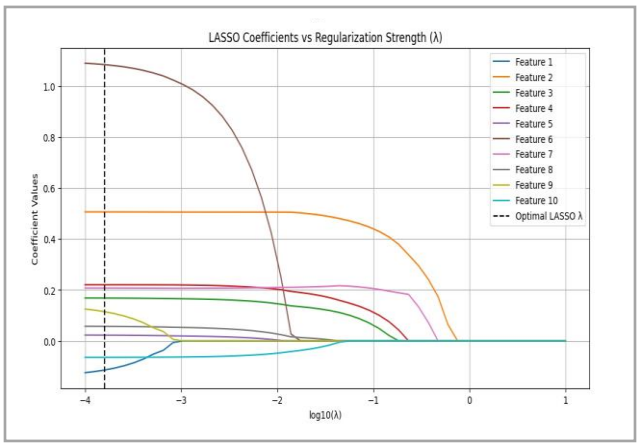
Performance Scores		
LASSO		Ridge
MSE	0.2924	0.2919
MAE	0.3052	0.3025
R^2	0.4535	0.4545

Table2: Performance Scores

This qraph used to understand difference between the models:







MSE vs. λ

Blue Line (LASSO):LASSO works well for small λ values (less penalty).

When λ increases, the error increases sharply because LASSO forces many features (variables) to zero.

Red Line (Ridge): Ridge is more stable and doesn't get much worse even with larger λ

R^2 VS λ

Blue Line (LASSO):

LASSO's R^2 drops a lot when λ increases, meaning it struggles to explain the data under strong regularization.

Red Line (Ridge): Ridge's R^2 stays high, showing it handles large penalties better.

SO we learn in this cases the Ridge regression is slightly better here because it handles regularization smoothly without making the model unstable.

LASSO regression is useful when you want to reduce the number of features (some coefficients become zero), but it's more sensitive to large λ

→ Ridge is the better choice for accurate predictions.

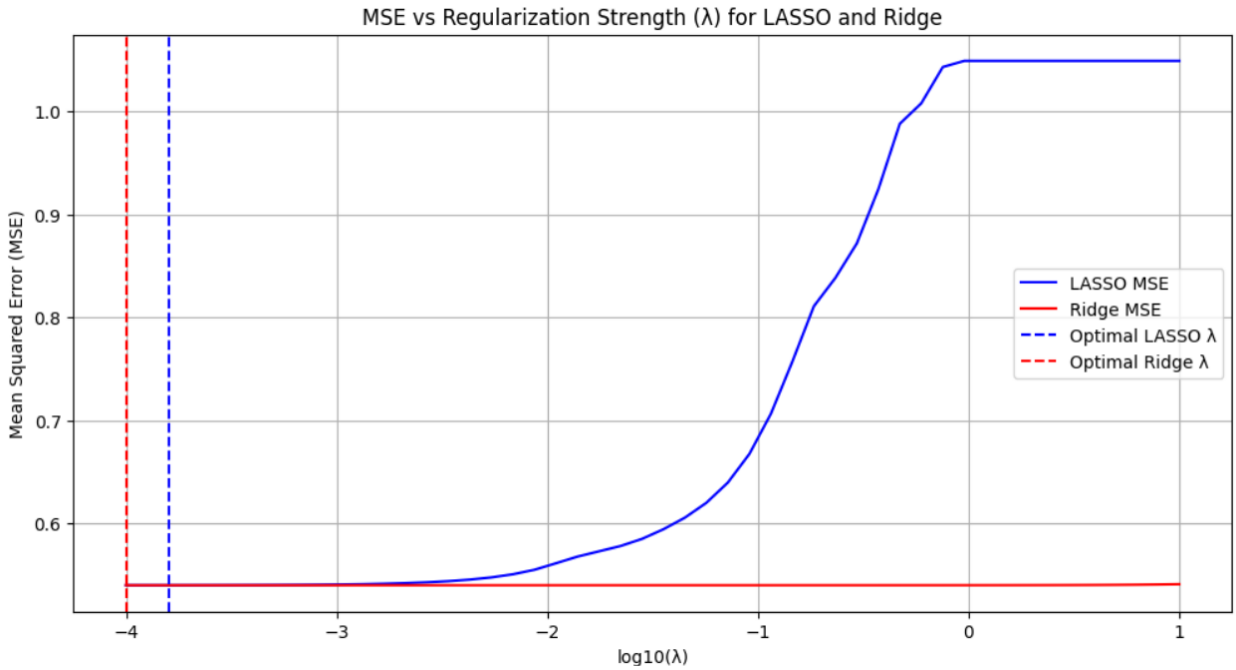
Part6: Hyperparameter Tuning with Grid Search

We are comparing three models: LASSO regression, Ridge regression, and Random Forest regression. Each model was tuned using Grid Search to find the best parameters that give the lowest error and highest accuracy.

The result:

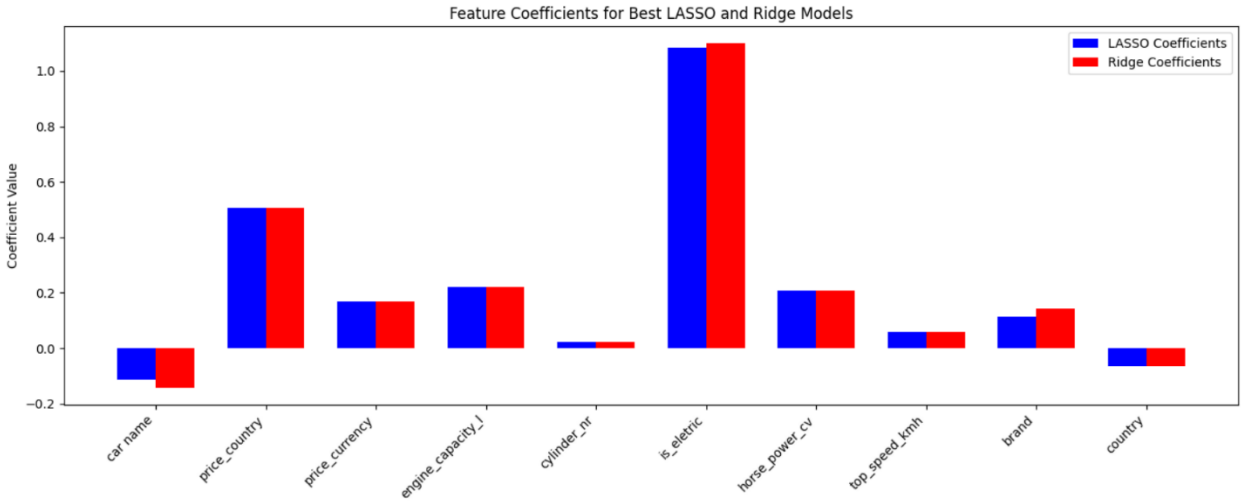
```
LASSO Grid Search:
Best λ: 0.00015998587196060574
MSE: 0.2924, MAE: 0.3052, R-squared: 0.4535

Ridge Grid Search:
Best λ: 0.0001
MSE: 0.2926, MAE: 0.3057, R-squared: 0.4532

Random Forest Grid Search:
Best Parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
MSE: 0.0159, MAE: 0.0488, R-squared: 0.9703
```

The figure shows result three models: LASSO regression, Ridge regression, and Random Forest



The figure shows MSE vs Regulation strength for lasso and Ridge

The dashed vertical lines represent the optimal λ values for LASSO (blue) and Ridge (red).



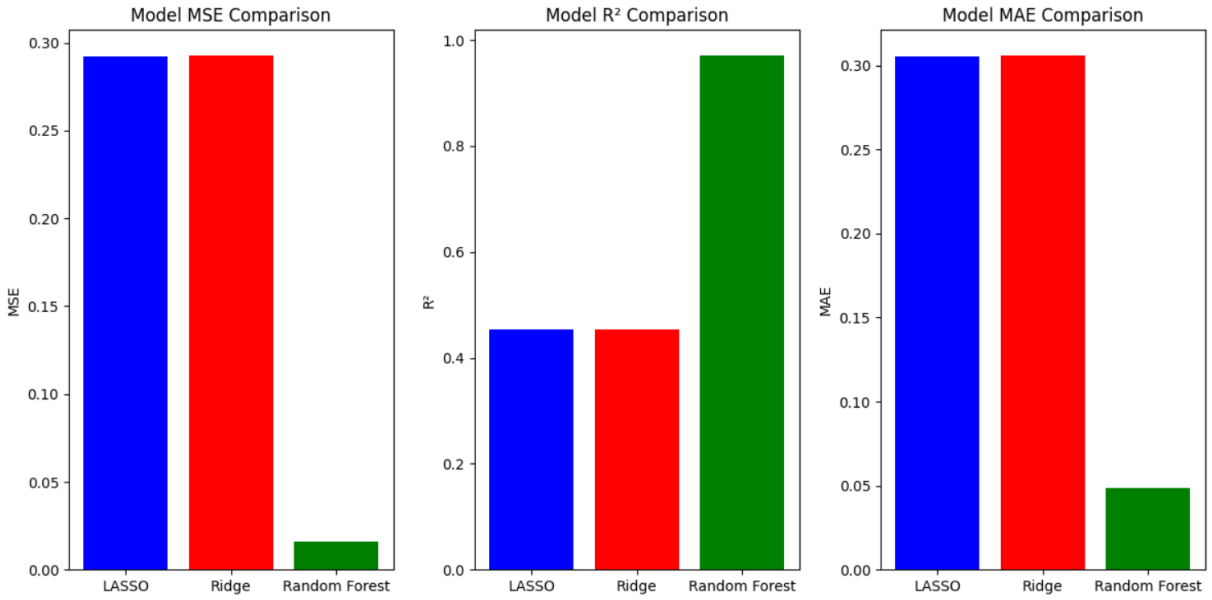
The figure shows Feature Coefficients for Best LASSO and Ridge Models.

LASSO Regression: Some coefficients are exactly 0. This happens because LASSO performs feature selection by shrinking less important features to 0.

Ridge Regression: All coefficients are non-zero, even if some are very small. This is because Ridge does not eliminate features but reduces their impact instead.

Features like "horse_power_cv" and "is_electric" have high coefficients for both models, indicating they are very important for predictions.

Less important features like "cylinder_nr" and "top_speed_kmh" have smaller coefficients in both models.



Random Forest appears to be the best-performing model among the three based on all three metrics (lowest MSE and MAE, highest R^2). LASSO and Ridge show comparable results but are less effective than Random Forest.

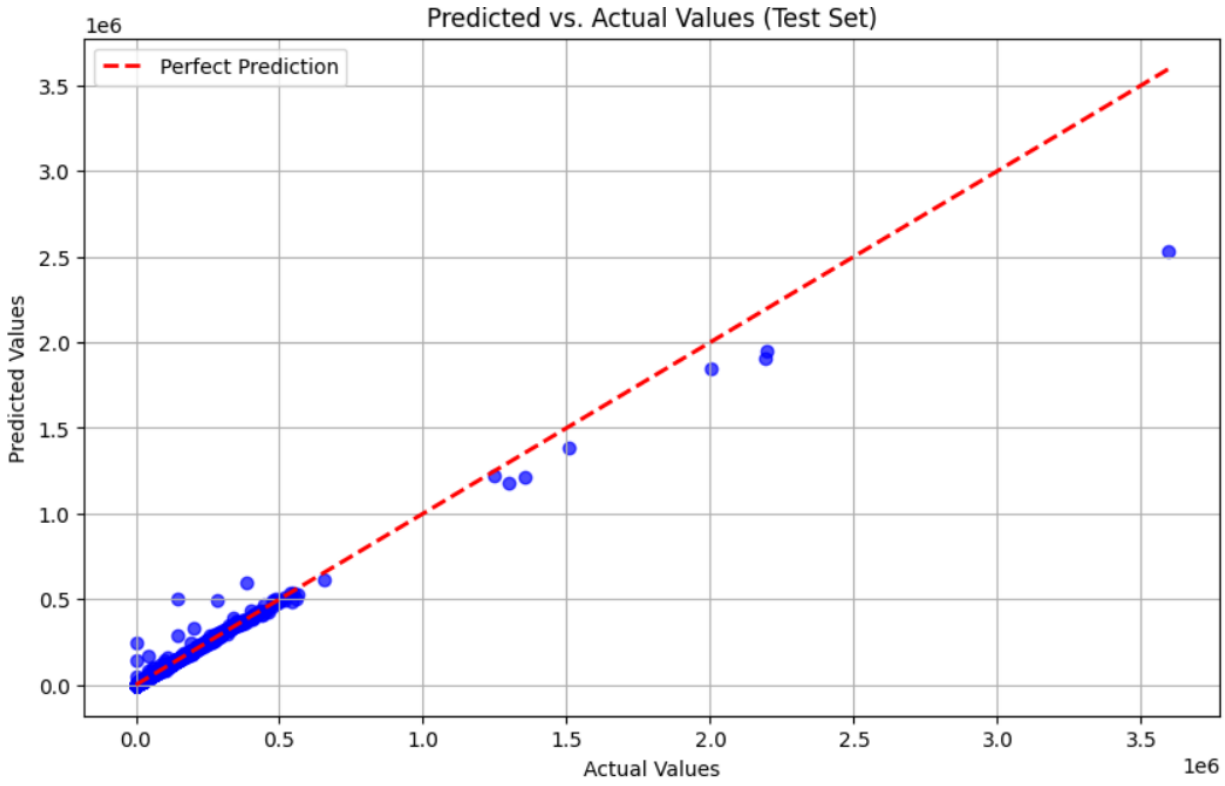
Part7: Model Evaluation on Test Set

We make Model Evaluation Metrics :

The result

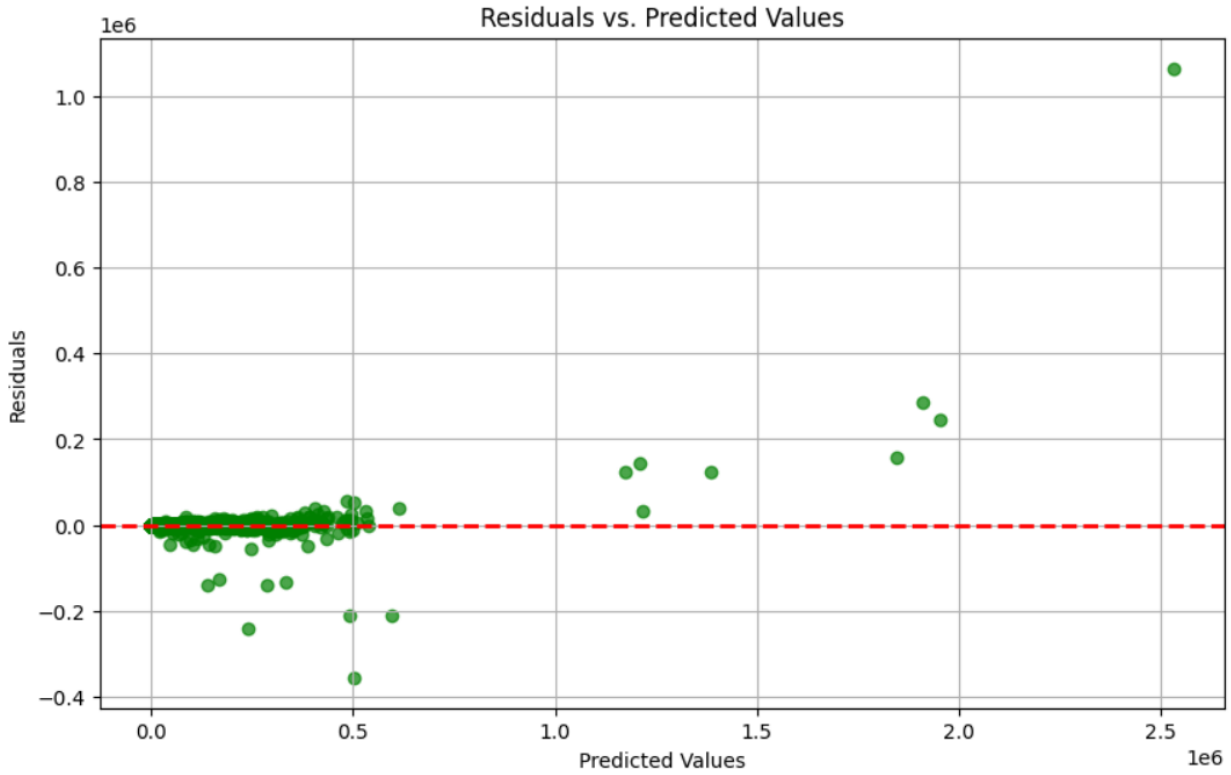
```
Model Evaluation on Test Set:
Test MSE: 279261258.8471
Test MAE: 1327.5610
Test R-squared: 0.9712

Interpretation:
The chosen model achieves a test MSE of 279261258.8471, showing the average squared prediction error.
The test MAE of 1327.5610 reflects the average absolute prediction error.
The R-squared value of 0.9712 reflects the proportion of variance in the test data explained by the model.
```

The result as scatter plot :

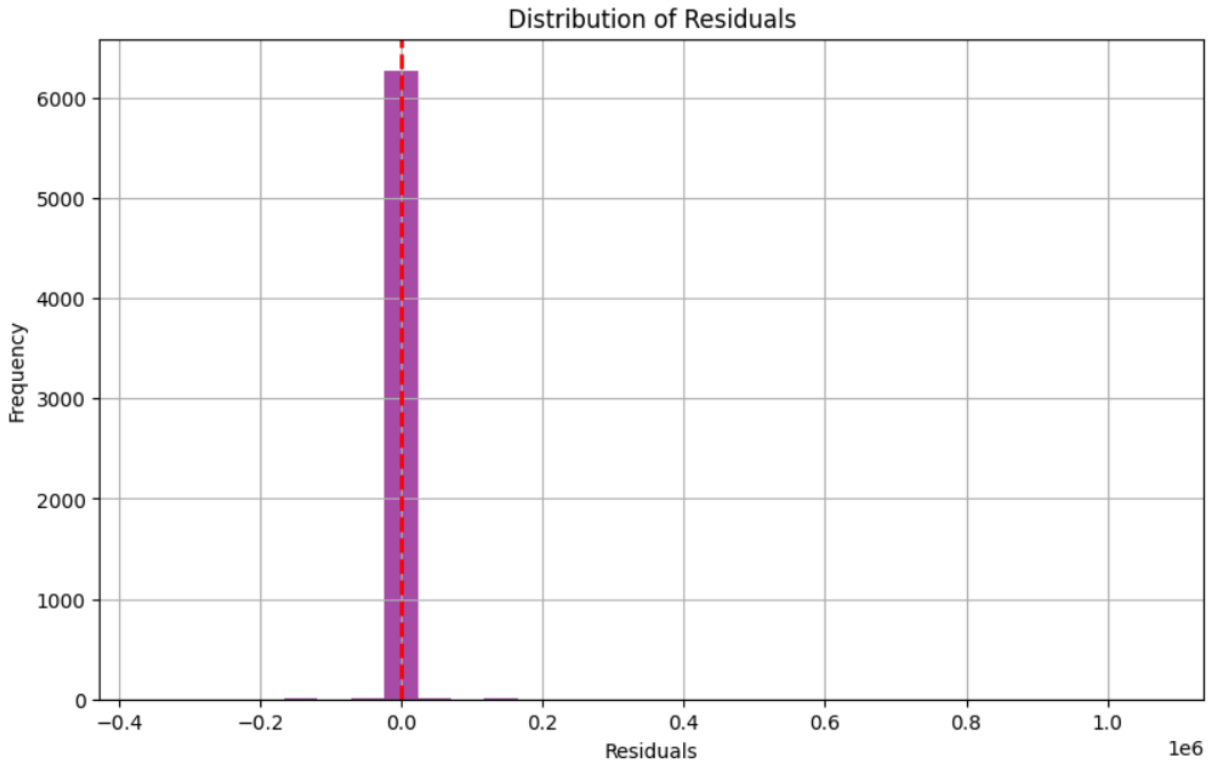


Most points lie close to the red line, indicating that the model performs well in predicting the target values.

A few points deviate from the line, suggesting slight inaccuracies in certain predictions, but overall, the alignment is strong



The residual plot shows that the residuals are not randomly distributed, with an increasing spread as predicted values grow, indicating potential heteroscedasticity. While the model performs well for lower predicted values, it struggles with higher predictions, leading to larger errors and outliers. This suggests the model might be missing key variables, or its assumptions (like linearity or constant variance) may not hold. Improving the model could involve adding more variables, applying transformations, or exploring non-linear methods.

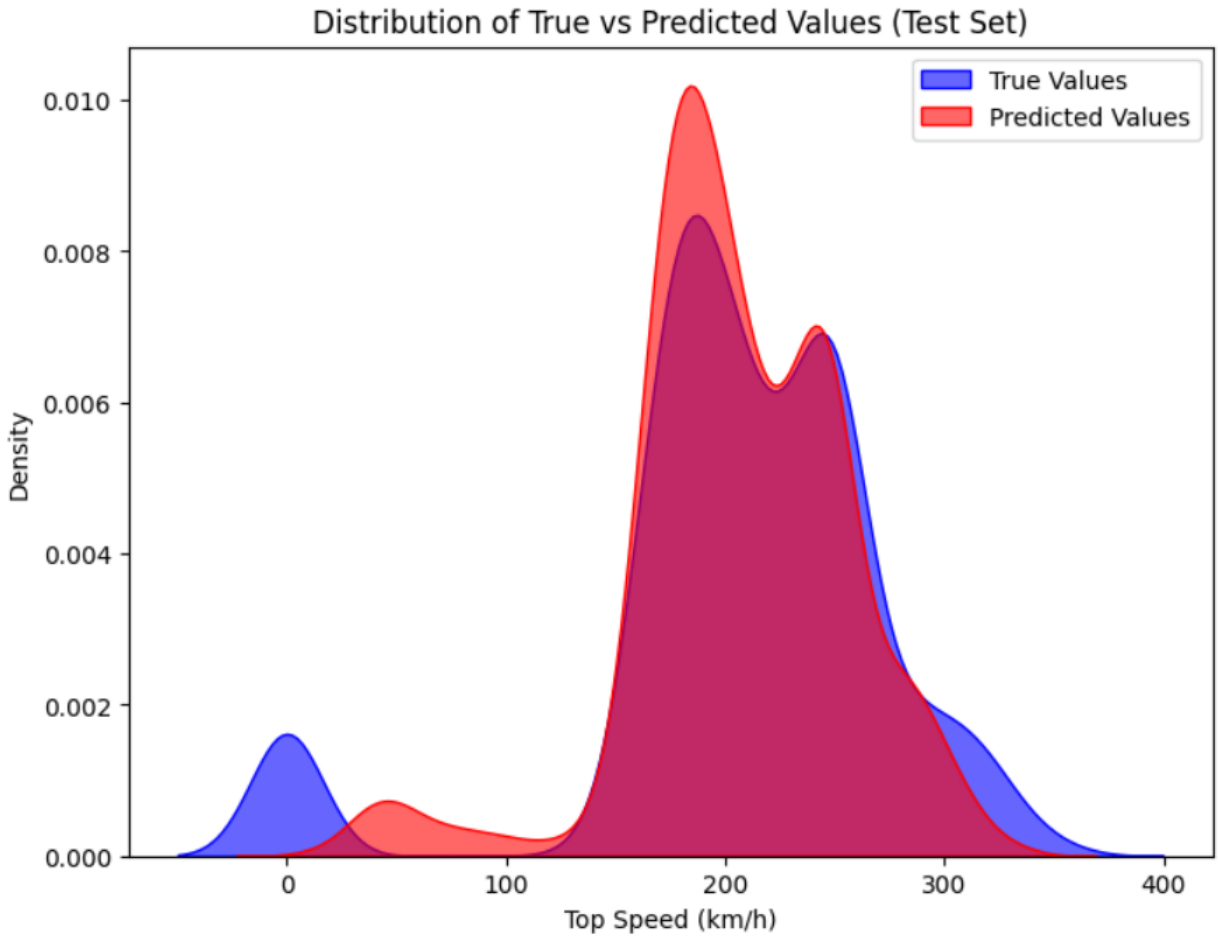


This histogram shows the distribution of residuals (errors) in the model. Most residuals are concentrated around zero, indicating that the model predicts well for the majority of data points. The red dashed line at zero suggests the ideal error value, and the distribution is symmetric around it. However, the presence of extreme values (outliers) far from zero indicates that the model struggles with certain predictions. This could suggest the need for further refinement of the model, such as addressing outliers or improving feature selection.

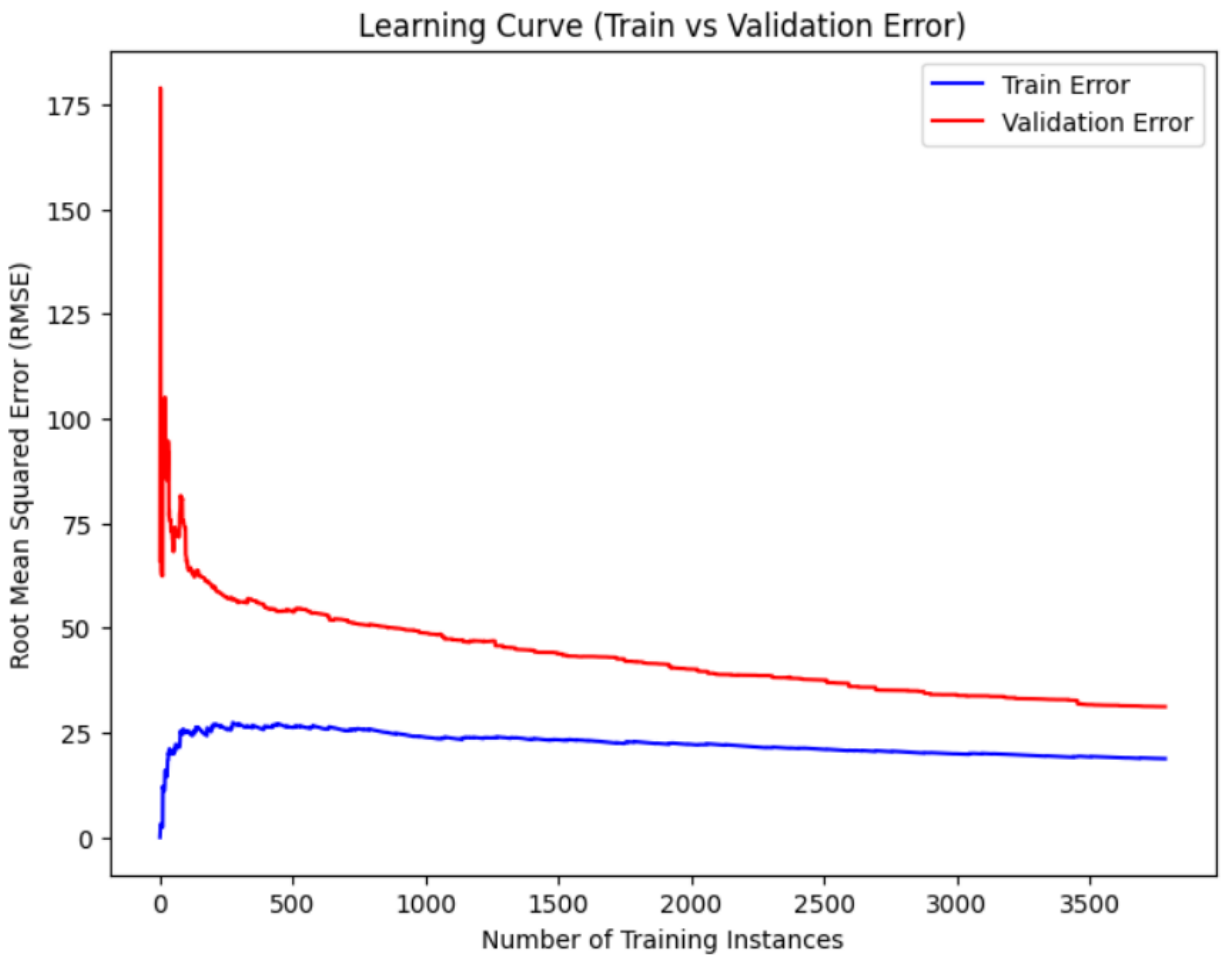
Part8 : identifying another relevant target variable in the dataset

In this part we make comparasion:

We are choose the '**top speed**' as another relevant target variable in the dataset



The density plot compares the distribution of true values (blue) and predicted values (red) for the test set. There is significant overlap between the two distributions, indicating that the model performs well in capturing the general trend of the data. However, some discrepancies are noticeable, particularly at the lower range and around the peak, where the true and predicted values diverge slightly. This suggests that while the model is generally accurate, it struggles with certain ranges of the data. Refinements, such as better feature engineering or model adjustments, could help address these deviations and improve performance.



This graph shows the representation of the relationship between the number of training instances (on the x-axis) and the Root Mean Squared Error (RMSE) for both training and validation data (on the y-axis). The blue curve represents the training error, while the red curve represents the validation error. Initially, the training error is low, but as the dataset size increases, the training error stabilizes at a higher value. The validation error starts high, indicating poor performance, but decreases as more data is used for training, eventually converging toward the training error. The consistent gap between the two curves suggests some level of bias or underfitting. However, the declining validation error indicates that the model improves as more training data is added.