Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Machine Learning ENCS5341

Assignment #2

Prepared by:

Hiba Jaouni 1201154

Amany Hmidan 1200255

Instructor: Dr. Ismail Khater

Section: 3

Date: November 2024

Introduction

This assignment involves building and evaluating regression models to predict car prices using a dataset from YallaMotors. The task includes data preprocessing, exploratory data analysis (EDA), and the application of both linear and nonlinear regression techniques, including LASSO, Ridge, and polynomial regression.

The code was written and executed using Google Colab. Key techniques such as feature selection, regularization, and hyperparameter tuning were employed to improve model accuracy and prevent overfitting.

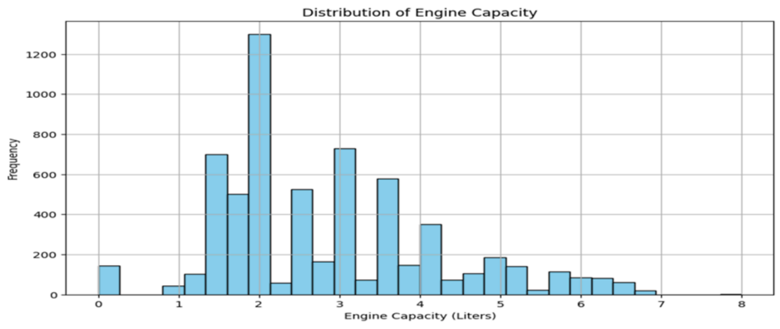
Dataset Exploration and Preprocessing:

The dataset consists of 6,308 entries and 9 columns, collected from YallaMotors for predicting car prices. All features are categorical, with columns such as car name, price, engine capacity, cylinder, horse power, top speed, seats, brand, and country.

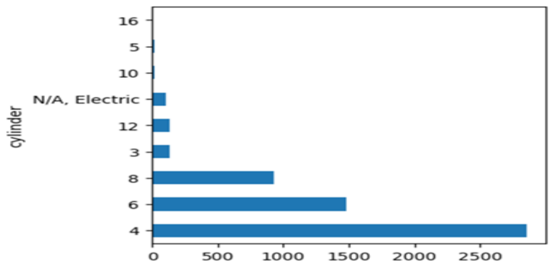
The dataset contains missing values in the 'cylinder' column, which has 5684 non-null entries, while other columns are complete. Given that all features are of object type, they require preprocessing and cleaning to prepare for analysis and modeling.

The engine_capacity feature exhibits inconsistencies and potential errors in its data. Some entries, such as 'Cylinders', '140', or '5200', do not represent valid car engine capacities. Additionally, the presence of numeric values like 'Cylinders' suggests data quality issues. To address this, we normalized the units by assuming that values greater than 10 represent cubic centimeters (cc) and converted them to liters by dividing by 1000.

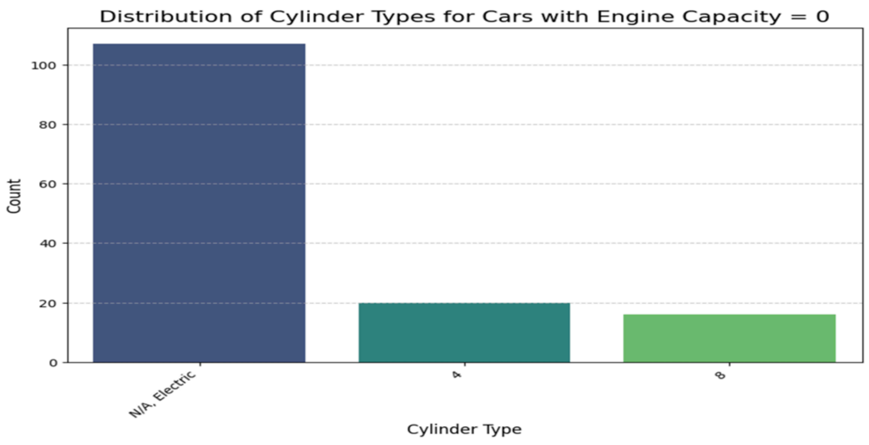
Furthermore, it was observed that 143 cars have an engine capacity equal to 0. From a practical standpoint, an engine capacity of zero is unrealistic for any functional car, indicating that these values may represent missing or erroneous data. This issue needs to be handled during preprocessing to ensure the dataset reflects realistic and valid car specifications.



The cylinder feature has 624 missing values, representing 1.11% of the total data. Upon examining the distribution of cylinder types, it is evident that most vehicles have 4 cylinders, followed by 6 and 8 cylinders. Additionally, some entries are labeled as "N/A Electric," indicating electric vehicles. For cars with an engine capacity of 0, "N/A Electric" reliably signifies an electric car.

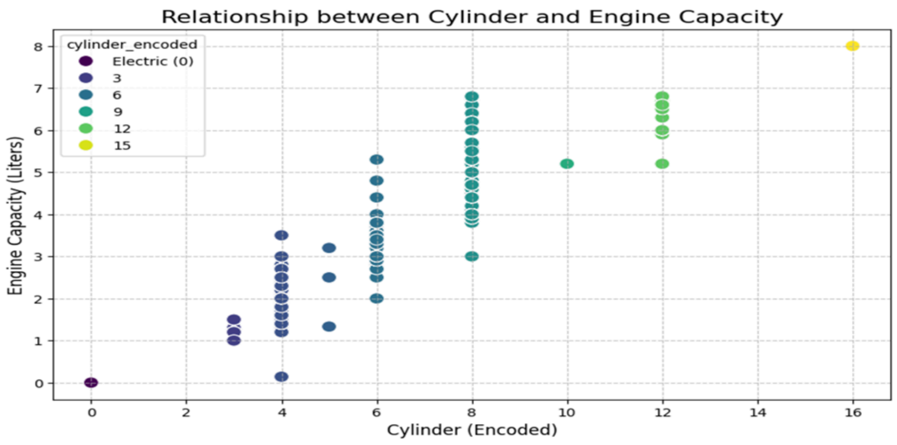


The analysis shows that most cars with an engine capacity of 0 are indeed electric, making this classification practical. However, cars with both engine capacity equal to 0 and cylinder types of 4 or 8 are considered outliers and were removed from the dataset. A total of 36 rows with engine capacity 0.0 and cylinder values of 4 or 8 were dropped, reducing the dataset size to 6,269 rows.



To handle the 'N/A Electric' entries, ordinal encoding was applied. Each cylinder type was encoded with its respective number, while 'N/A Electric' was assigned a value of 0 to distinguish electric cars from those with internal combustion engines.

To fill the missing values in the cylinder feature, the correlation matrix revealed a high correlation of 0.93 between engine_capacity and cylinder, indicating that these two features are strongly related. Given this relationship, the missing values in the cylinder feature will be imputed based on the corresponding values in the engine_capacity feature.



The following conditions were defined to fill the missing cylinder values based on the engine_capacity_cleaned values:

1. **Engine capacity = 0**: The vehicle is an electric car, so the cylinder is set to 'N/A Electric'.

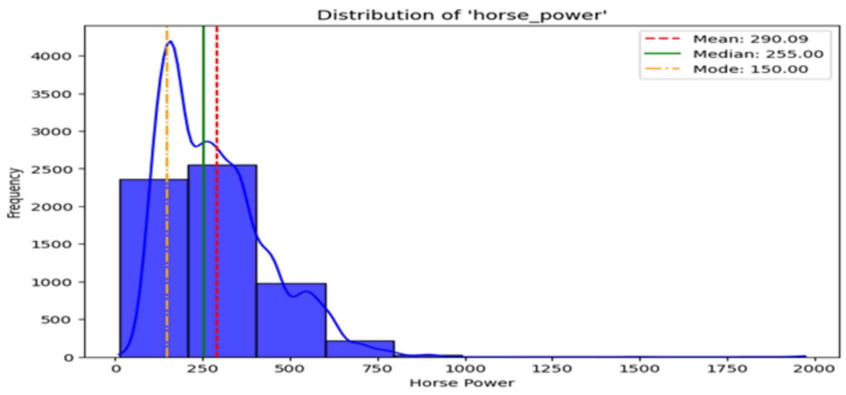
2. **0 < Engine capacity <= 1.5**: The vehicle likely has a smaller engine, so the cylinder value is set to 4.
3. **1.5 < Engine capacity <= 3**: The cylinder value is set to 4, reflecting typical mid-sized engines.
4. **3 < Engine capacity <= 4.5**: The vehicle is likely to have 6 cylinders, so the cylinder value is set to 6.
5. **4.5 < Engine capacity <= 6**: The cylinder value is set to 6, representing larger engines.
6. **6 < Engine capacity <= 7**: The cylinder value is set to 8, typically associated with larger engines.
7. **7 < Engine capacity <= 8**: The cylinder value is set to 8 for even larger engine sizes.

By applying these conditions, the missing values in the cylinder column can be imputed accurately based on the corresponding engine capacity, ensuring consistency and leveraging the high correlation between these features.

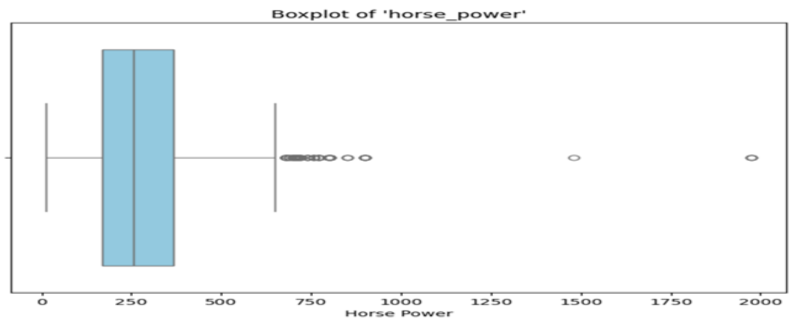
The horse_power feature, which should typically contain numeric values, includes non-numeric entries like 'Single', 'Double', and 'Triple'. These non-numeric values likely represent categorical data but are not appropriate for a numerical feature. Additionally, some numeric values such as '1973' and '1479' appear unusually high for horsepower, suggesting data entry errors or invalid records.

To address this, the horse_power column was converted to a numeric type using pandas.to_numeric(). Non-convertible entries (like 'Single', 'Double', 'Triple') were replaced with NaN. After this conversion, the mean, median, and mode were computed:

- **Mean of 'horse_power'**: 290.09
- **Median of 'horse_power'**: 255.00
- **Mode of 'horse_power'**: 150.00

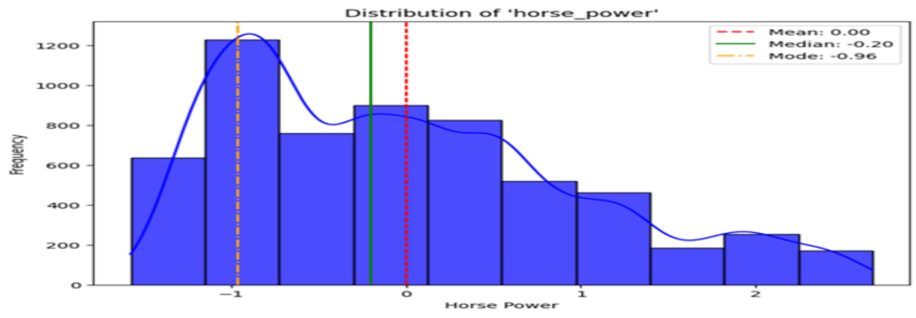


Next, a box plot and percentiles were used to identify outliers, with 116 outliers detected, accounting for 1.89% of the data. The presence of outliers can significantly skew the mean, as seen with the mean of 290.09, which is much higher than the median and mode, suggesting extreme values.



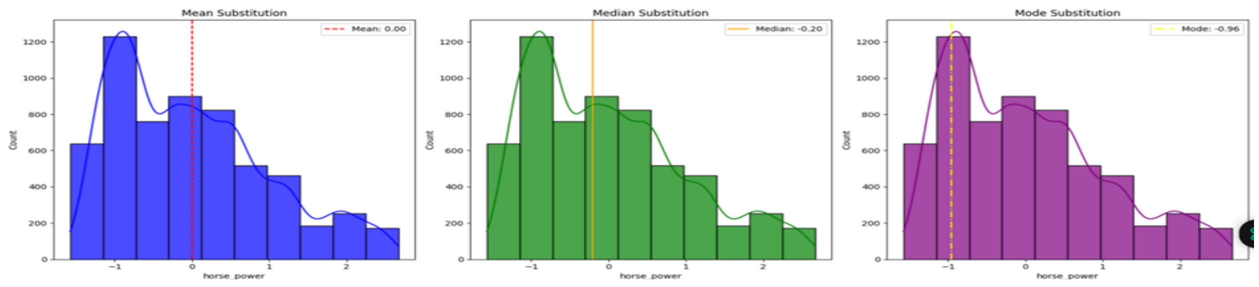
To handle the outliers, the Interquartile Range (IQR) method was applied. After removing the outliers, the updated statistics were:

- **Mean of 'horse_power'**: 0.0
- **Median of 'horse_power'**: -0.20
- **Mode of 'horse_power'**: 0.96

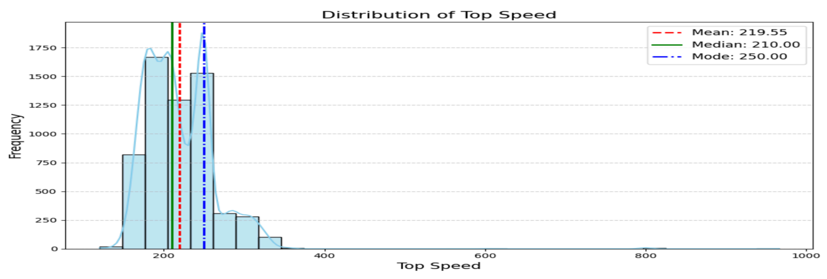


Removing outliers has reduced the skewness in the distribution, bringing the mean closer to the median and mode, providing a more accurate representation of typical vehicle horsepower.

To handle missing values in the horse_power feature after conversion to numeric values, we considered mean, median, and mode substitution. Since the statistics for all three methods were identical, indicating a low proportion of missing values, the missing values were filled using the **median.**



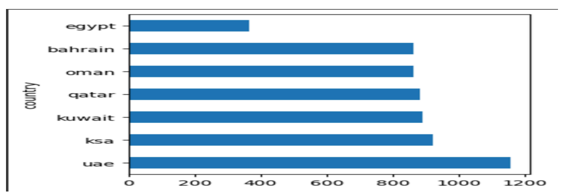
The top_speed feature included non-numeric values like "8 Seater" and "14 Seater," reflecting inconsistent data. These values were converted to NaN using pandas.to_numeric() to clean the column. The median of the valid numeric entries was calculated and used to replace the missing NaN values, ensuring the feature remained consistent and fully numeric while maintaining its overall distribution.



The seats feature contained inconsistent values, with 93.83% of rows including the keyword "Seater," which represents valid data (the majority of data), and 4.61% having numeric values between 2 and 30, a reasonable range for the number of seats. To clean the data, the numeric part of strings like "4 Seater" was extracted and converted to integers. Values with decimals, such as 2.2 or 25.8, were rounded down to the nearest integer (e.g., 2 and 25, respectively). Invalid entries (representing a small portion of data), such as "Automatic" or values outside the range of 2 to 30, were explicitly replaced with NaN.

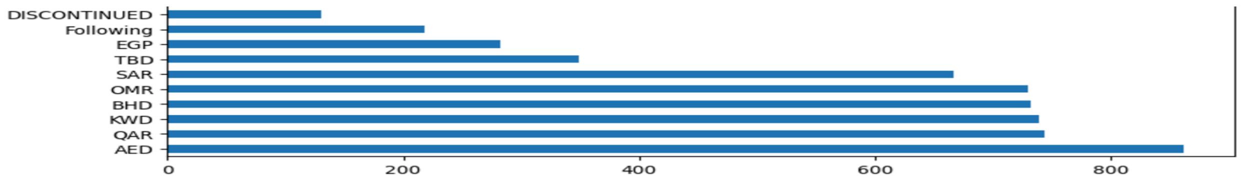
The seats column was updated directly in the original dataset, and rows with NaN values were dropped to ensure only valid data remained. After cleaning, the distinct values in the seats column were reduced to integers within the range, leaving consistent and meaningful data for analysis and, at the same time, the dropped rows did not cause much effect since they accounted for less than 2%.

The brand, country, and car name features were found to be clean and consistent upon reviewing their distinct values. These columns did not contain any irregular or invalid entries and appeared rational, requiring no further cleaning.



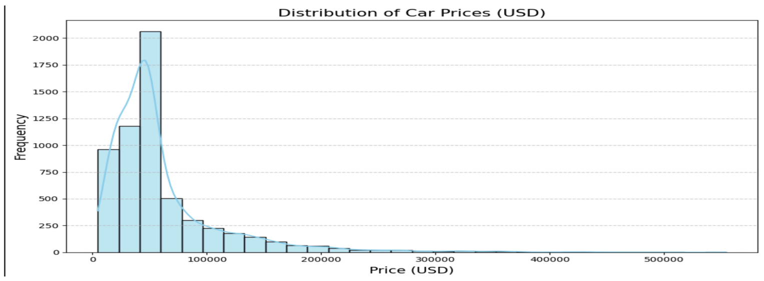
The price feature was cleaned by converting values to USD using predefined currency conversion rates and extracting numeric data from strings. Invalid entries, which represented 19.93% of the data, were replaced with the median price instead of being dropped. Dropping nearly 20% of rows could introduce bias or significantly reduce the dataset size, which might affect model performance.

The currencies that were used in the conversion in the previous step are SAR, AED, OMR, QAR, KWD, BHD, and EGP which is practical to consider since they represent the currencies for the 7 countries that exist in the dataset: Saudi Arabia, Bahrain, Kuwait, United Arab Emertes, Qatar, Oman and Egypt.

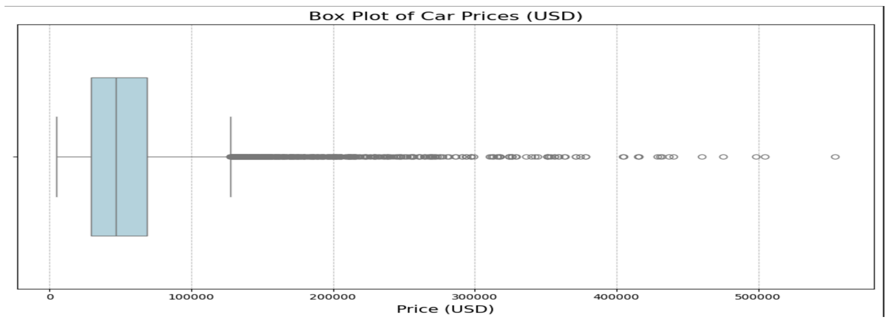


The median was selected for imputation instead of the mean or mode because it is less sensitive to extreme values (outliers) in the data. While the mean price ($64,841.33) was skewed higher by a few expensive cars, the median price ($46,437.38) was considered a better representation of the central tendency of the typical data. The mode ($26,080.00), although frequent, did not capture the overall distribution as accurately as the median. By using the median, a more balanced and representative imputation for the missing values was ensured.

The price feature may have outliers based on the blow figure which shows its distribution.



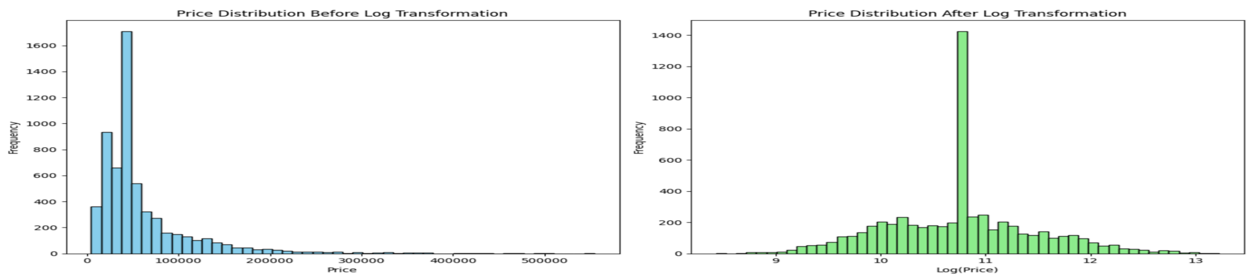
The boxplot revealed that 5.00% of the prices were above the 95th percentile, indicating the presence of outliers in the data. These outliers likely represent higher-priced vehicles, which could include luxury cars or rare models. Although these values are extreme compared to the majority of the dataset, they were left unchanged during the cleaning process. This decision was made because high-priced vehicles are still valid data points that reflect the market for luxury cars, which should not be disregarded. Removing these outliers could result in losing valuable information about this segment of the market.



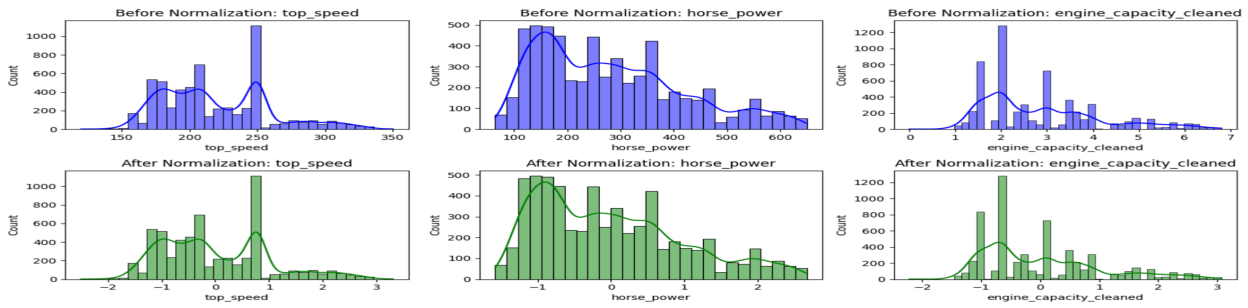
Normalization and Encoding:

For the target variable, normalization or transformation improves the performance of some models, especially linear models (such as Linear Regression, LASSO, or Ridge) and neural networks. These models tend to perform better when both the target variable and features are on similar scales, helping to reduce bias in model training and ensuring faster convergence.

Given that the price distribution is positively skewed and may contain extreme outliers, a log transformation was applied to the price data. This transformation helps to normalize the distribution, making it closer to a normal distribution and reducing the impact of outliers.



For the features top_speed, horse_power and engine_capacity, Z-score normalization was performed. Z-score normalization is particularly useful for regression models, as it ensures that all features are on the same scale, preventing models from being biased toward variables with larger magnitudes. By transforming these features, the influence of each feature on the regression model becomes more comparable, improving the model's ability to learn the relationships between the input variables and the target.



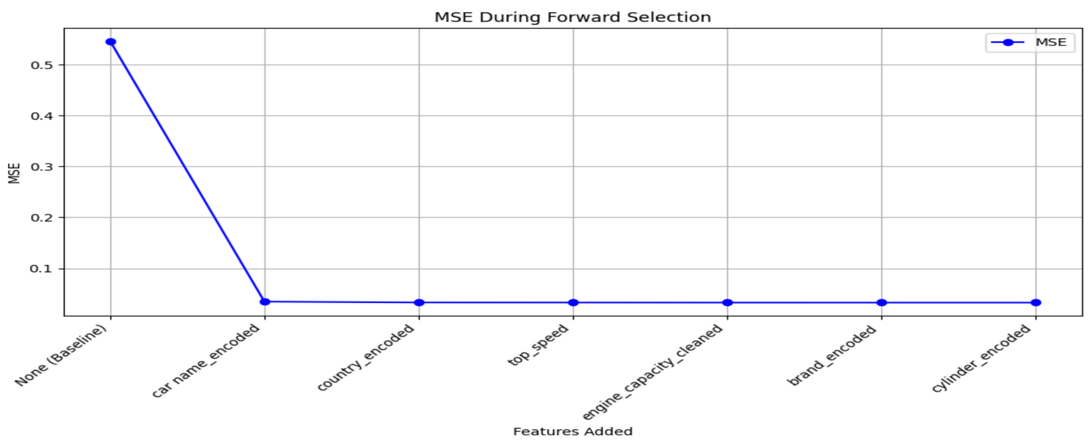
Label Encoding was applied to the features of car name, brand, and country to convert categorical data into numerical values for regression models. While effective, care must be taken to ensure the model doesn't treat these encoded values as ordinal, as they have no inherent order. One hot encoding is a bad option since it adds more columns and overhead and reduces performance, especially in higher dimensional spaces.

After Model training, we decided to use target encoding for car name, brand, and country features since it improved model performance. It is explained better in the next section.

Linear Regression:

Linear Regression with Forward Selection

For the Linear Regression model, the dataset was split into three subsets: 60% for training, 20% for testing, and 20% for validation.
In order to build a linear regression model for our dataset, we used feature selection with the forward selection method, which is an iterative process to select the best subset of features for a regression model.



The baseline predicts the mean of the target variable (log_price) for all samples. Its performance metrics (MSE, MAE, R-squared) serve as a benchmark. Performance metrics at this point were as follows: MSE: 0.5463, MAE: 0.5443, R-squared: -0.0000 (No predictive power, as expected for a baseline).
As features like car name_encoded, country_encoded, and top_speed are added, the MSE drops significantly, showing that these features are very valuable for predicting the target variable. After adding engine_capacity_cleaned, brand_encoded, and cylinder_encoded, the MSE becomes very stable and very low, indicating that adding these features does not improve the model much further.

Results:
Best Model Selected:

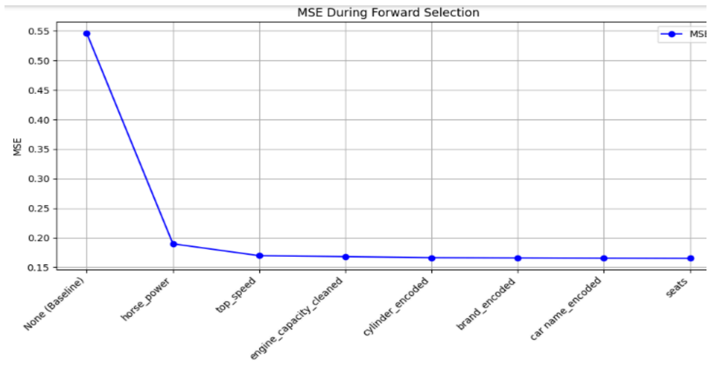
- **Selected Features:** ['car name_encoded', 'country_encoded', 'top_speed', 'engine_capacity_cleaned', 'brand_encoded', 'cylinder_encoded']
- **MSE:** 0.0324
- **MAE:** 0.1127
- **R-squared:** 0.9406

The MSE and MAE are low, and R-squared is high, indicating well performance and good generalization.

Note that two features weren't included in the model which are *horse_power* and *seats*. To look for the possible reason, we calculated the correlation between *seats* and *log_price*(target feature), and it was **0.233** which indicates a weak correlation suggesting that *seats* have a minimal relation with price. However, the correlation between *log_price* and *horse_power* was **0.79** indicates a strong positive correlation; so the reason why it wasn't selected is likely because its contribution was overshadowed by other features already in the model. The correlation between *horse_power* and *engine_capacity_cleaned* is 0.76, and between *horse_power* and *top_speed* is **0.7**, which are high, indicating that these features are capturing similar information, that's why *horse_power* wasn't included.

Note:

For the following features *car name, country, brand*, target encoding was used. However, we previously tried using label encoding, and the results were as following:



Notice that the error is larger, and the importance of features in the graph are different. That's because this encoding method simply assigns an integer to each category. This approach can introduce an implicit ordinal relationship, which may mislead the model into interpreting these integers as meaningful numeric values.

On the other hand, with Target Encoding , the categories are replaced by average prices. This provides a more meaningful encoding for categorical variables by directly relating them to the target, which can improve feature importance ranking, especially for variables like car name and brand, and resulted in a smaller error.

Closed- Form Solution

In this part, we implemented closed form solution to find weights from this equation:

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

No external APIs were used, only numpy library to help with calculations.

Here are the result:

Closed-form solution weights: [-5.46314978e+01 -6.79122014e-03 1.49810258e-02 8.08065857e-04

1.35507576e-02 1.39986865e-03 5.69490832e+00 2.25311876e-02

3.61330303e-01]

Then we calculated weights by using the gradient descent method, which iteratively adjusts the weights to minimize the cost function. The initial weights were set randomly, and then the model updated them step-by-step based on the gradients of the cost function.
Results:
```
Iteration 0, Cost: 15.098459064028884
Iteration 100, Cost: 3.802583220008317
Iteration 200, Cost: 2.421486761184384
Iteration 300, Cost: 1.766652706152056
Iteration 400, Cost: 1.377573960203939
Iteration 500, Cost: 1.1196016188619287
Iteration 600, Cost: 0.939635853050176
Iteration 700, Cost: 0.810383444963566
Iteration 800, Cost: 0.7153882638953901
Iteration 900, Cost: 0.6439634957860062
Gradient Descent weights: [-0.13955116 -0.09367282  0.67561411 -0.051951
-0.64232777  0.50988951
  1.70110698 -1.03370131  0.14149882]
```

The weights obtained from both the closed-form solution and gradient descent are reasonable and show expected relationships between features and the target variable (log_price), if the gradient descent weights had been significantly different from the closed-form solution, it could have indicated issues such as incorrect scaling, too large a learning rate, or insufficient iterations, but they are similar.

Linear Regression with LASSO and Search Grid

The dataset was used to train a Lasso regression model with log_price as the target and selected features. A Grid Search was performed to identify the optimal alpha (λ) by testing a range of values. For smaller alphas (from 1e-5 to 0.0001265), the MSE remained stable around 0.0315, indicating good performance. However, as alpha increased past 0.0001265, the MSE started rising, suggesting stronger regularization led to underfitting.

The best alpha was **0.0001265**, yielding an MSE of 0.0324, MAE of 0.1130, and R-squared of 0.9406. Smaller alphas allow better model fitting, while larger alphas cause underfitting. Beyond **0.0001265**, increasing alpha worsens the model's performance (MSE rises to 0.4973). Thus, an alpha near **0.0001265** strikes the optimal balance between overfitting and underfitting, ensuring the best model performance.

For Lasso with **alpha = 1e-05**, five features were selected with non-zero weights: ['horse_power', 'top_speed', 'seats', 'brand_encoded', 'cylinder_encoded'], indicating their significant contribution to the model. The features ['car name_encoded', 'country_encoded', 'engine_capacity_cleaned'] were assigned zero weights, suggesting they had little to no impact on the model's performance. Feature selection is a feature that distinguishes LASSO from Ridge.

Linear Regression with Ridge and Search Grid

The Ridge regression model was trained with log_price as the target and selected features. A Grid Search was conducted to find the best alpha (λ), and the results showed that for smaller alpha values (1e-5 to 0.0115), the MSE remained stable around 0.0315. However, as alpha increased beyond 0.0115, the MSE started to rise, indicating underfitting.

Best Alpha (λ): The optimal alpha was **0.0115**, with an MSE of 0.0325, MAE of 0.1128, and R-squared of 0.9406. Smaller alphas allow better fitting, while larger alphas lead to underfitting. The best performance was achieved at **0.0115**, balancing regularization and model fit.

Nonlinear Models:

Polynomial Regression

In this part we will apply polynomial regression with degrees varying from 2 to 10. Polynomial regression allows us to model nonlinear relationships between features and target.

First, we built the non-linear models using the selected features from the forward selection process, to prevent including irrelevant features.

Results using selected features:
Polynomial Regression Model Evaluation Metrics:

Degree	**MSE**	**MAE**	**R-squared**
2	1.564188e-01	0.312922	7.136486e-01
3	1.423186e-01	0.293987	7.394615e-01
4	1.330567e-01	0.281787	7.564170e-01
5	1.358595e-01	0.278611	7.512859e-01
6	7.078595e-01	0.288324	-2.958575e-01
7	1.200576e+01	0.379374	-2.097859e+01
8	2.491592e+03	2.254899	-4.560284e+03

9	4.210567e+04	10.565260	-7.708061e+04
10	1.271310e+07	116.757364	-2.327349e+07

Table 1: Polynomial results with selected features

From the table above, we observe that for degrees 2 to 5, the MSE and MAE decrease, and the R-squared improves. This suggests that the model is getting better at fitting the data with increasing polynomial complexity.
On the other hand, Degrees 6 to 10 are likely overfitting the data, as indicated by the rapid increase in MSE and the drastic decrease in R-squared.

Second, we included all features in the training, to evaluate all features with polynomial regression.

Results using all features:

Polynomial Regression Model Evaluation Metrics:

Degree	**MSE**	**MAE**	**R-squared**
2	3.334016e-02	0.112668	9.389652e-01
3	3.787777e-02	0.106121	9.306583e-01
4	5.152208e-02	0.115721	9.056801e-01
5	7.409451e+00	0.326258	-1.256426e+01
6	2.159088e+12	44570.4067	-3.952578e+12
7	2.507892e+14	481824.068	-4.591124e+14
8	1.765590e+13	154160.128	-3.232213e+13
9	2.931830e+12	86712.8827	-5.367215e+12
10	3.254851e+12	93067.0779	-5.958559e+12

Table 2: Polynomial results with all features

From the table above, we observe that for degrees 2 and 3, the polynomial regression with all features gives results similar to the linear model because the added complexity isn't necessarily improving performance, and it doesn't capture much beyond what the linear model does.
However, for degrees 5 and above: The model suffers from overfitting, which causes the dramatic increase in MSE, MAE, and negative R-squared values. This is expected when using high-degree polynomial models without any regularization.

Results analyzation:

We noticed that for degrees 2, 3, and 4, the polynomial regression with all features performs better (i.e., has lower error) than the polynomial regression with selected features. This was unexpected because typically, feature selection should reduce complexity and improve performance by removing irrelevant predictors. One reason for this could be that the selected features from the linear model might not be the best subset for polynomial regression, as feature selection is optimized for linear models and may not consider interactions or higher-order terms. Additionally, polynomial features can capture interactions between variables that are not reflected in the original linear features, so removing certain features may eliminate useful interactions.

However, even though for degrees 2, 3, and 4, the polynomial regression with all features performs better than with selected features, for degrees higher than 5, the error of using all features was much worse which indicates more overfitting when including more features.

Another observation that caught us thinking is that the results of polynomial models aren't much better than the linear model, this suggests that the relationship between the features and the target variable is almost linear or close enough to linear that a linear model can capture most of the underlying pattern.

Radial Basis Function (RBF) kernel model

In this part, the training of the model will depend on the Support Vector Machine (SVM) with the Radial Basis Function (RBF) kernel, which is mathematically the same as the Gaussian kernel. It is a powerful supervised learning algorithm that maps input features to a target value, with the goal of minimizing prediction errors. The RBF kernel allows the model to handle non-linear relationships between features and the target by mapping the data into a higher-dimensional space.

Key Hyperparameters for SVR with RBF Kernel:

C (Regularization Parameter): Controls the trade-off between achieving a low training error and a low testing error (bias-variance trade-off).

Gamma (Kernel Coefficient):Defines how far the influence of a single training example reaches.

Kernel: Determines the function used to map the data into a higher-dimensional space.

HyperParameter Tuning:

We used grid search to find the best parameters that give best performance with least errors.

These are the values we tried:
'C': [0.1, 1, 10, 100],
'gamma': [0.001, 0.01, 0.1, 1],
'kernel': ['rbf']
And here are the results:
Best parameters found: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Validation Set Performance:
MSE: 0.0360
MAE: 0.1027
R-squared: 0.9341

Test Set Performance:
MSE: 0.0289
MAE: 0.0970
R-squared: 0.9443

The model performs well on both the validation and test sets, suggesting that it has learned generalizable patterns rather than overfitting to the training data. The relatively low **MSE** and **MAE**, along with high **R²** values, suggest that the SVM with the chosen parameters (**C=1, gamma=0.1**) is a good model for the given data.

Evaluation Metrics on validation sets

	MSE	MAE	R-squared
Linear Model	0.0324	0.1127	0.9406
RBF Kernel Model	0.0360	0.1027	0.9341
Ridge	0.0115	0.0325	0.1128
LASSO	0.0324	0.0324	0.1130

For polynomial models, see evaluation results from Table1 and Table2 in previous sections.

Evaluation Metrics on test sets

	MSE	MAE	R-squared
Linear Model	0.0301	0.1105	0.9420

RBF Kernel Model	0.0289	0.0970	0.9443
Ridge	0.0302	0.1105	0.9418
LASSO	0.0302	0.1108	0.9419

Polynomial Regression (vary the polynomial degree from 2 to 10):
<u>With selected features:</u>

Degree	Test MSE	Test MAE	Test R-squared
2	0.1723	0.3246	0.6684
3	0.1647	0.3117	0.68299
4	0.2523	0.3075	0.51437
5	0.9066	0.3124	-0.74504
6	328.218	0.8144	-630.736
7	81135.6	8.6154	-156164.4
8	1449322000	1107.515	-2789577000
9	8193720000	2635.547	-15770830000
10	68887500000	7659.631	-1.32591E+11

<u>With all features:</u>

Degree	Test MSE	Test MAE	Test R-squared
2	0.032178	0.110651	0.938065
3	0.034531	0.103762	0.933536
4	0.086986	0.117349	0.832575
5	35.95509	0.591932	-68.20442
6	3.440305e+12	78,519.39	-6.621713e+12

7	1.868329e+15	1,348,246.00	-3.596059e+15
8	2.471839e+17	14,645,050.00	-4.757664e+17
9	2.801735e+16	5,021,537.00	-5.392630e+16
10	5.662107e+16	7,167,723.00	-1.089812e+17

In conclusion, the best model on test set was RBF Kernel Model.

For Further Details about the code, you can visit this site:
https://colab.research.google.com/drive/1Ly1zsRmoX81KlPBuvT10rgX0FrZjS5tl?usp=sharing