Used Car Price Prediction Using Machine Learning

Jingmin Xu
Data Science Institute
Date: 12/15/2024
Github

1 Introduction

1.1 Motivation

As a second-hand car owner, I am interested in predicting car price using machine learning since it can help understand what factors influence used car prices and avoid being misled by sales tactics. The used car market is complex, with prices influenced by factors such as make, model, age, mileage, and overall condition. By using machine learning, we can analyze extensive data sets to identify significant factors impacting prices more objectively.
As a result, both buyers and sellers can engage in transactions with greater confidence, supported by data-driven insights that ensure fair and transparent pricing. This project aims to leverage machine learning not only to enhance the accuracy of car valuations but also to empower consumers by demystifying the pricing process and reducing the potential for sales-driven price inflation.

1.2 Data Description

I found the dataset on Kaggle and it contains various features related to cars, including the year of manufacture, selling price, kilometers driven, fuel type, seller type, transmission type, number of previous owners, mileage, and engine specifications. The target variable of this dataset is the selling price (in local currency other than the United States), which is also the variable that we want to predict using machine learning. There are 12 columns in the original dataset.

2 Exploratory Data Analysis

There are 12 features in the dataset, and the total data points are 8128, with four continuous variables having missing values (Approximately 2.72%). I first plotted the histogram of the target variable, the selling price, to check the distribution. Noticing that there is a right-skewed and long right-tailed distribution, I applied the log transformation of the target variable to normalize it, which may further improve the model performance.

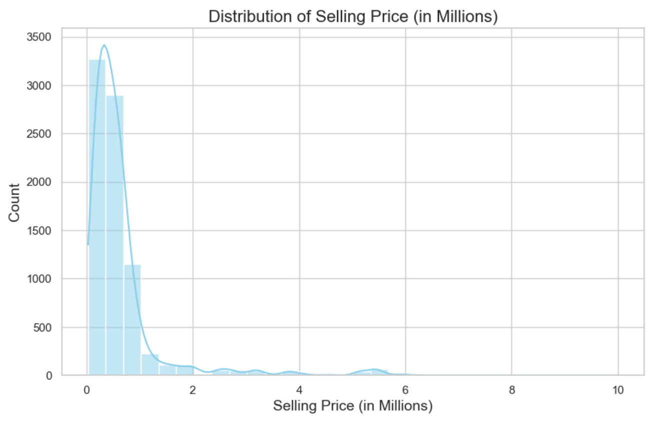


Fig 1. Distribution of Selling Price (in Millions)

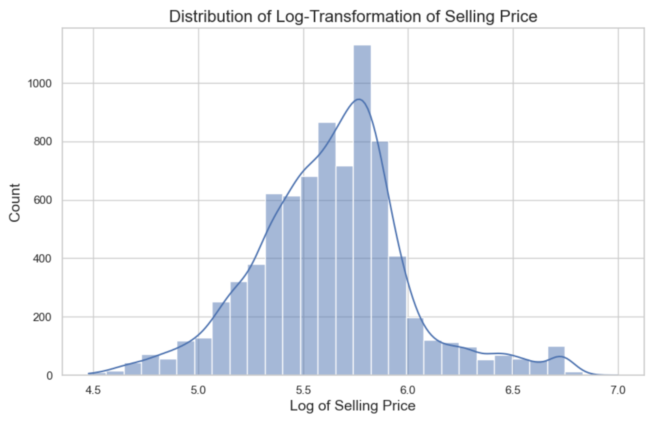


Fig 2. Distribution of Log-Transformation of Selling Price

I also plotted the scatter plot of the Selling Price versus the Year of Manufacture. From Fig 3, we see that newer cars, especially those manufactured closer to 2020, tend to have higher selling prices, while older models sell for less. However, there are exceptions, as some older cars also show relatively high prices, possibly due to collector value or excellent condition.

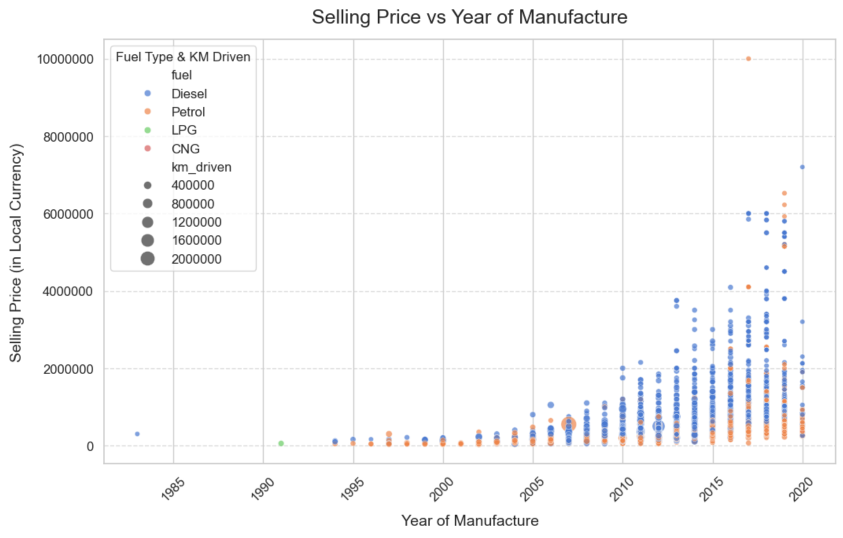


Fig 3. Selling Price vs Year of Manufacture

Moreover, I also plotted the box plot to check the Distribution of selling price by Owner (Fig 4). From this plot, there is a general depreciation that happens when adding a transfer of the cars. Also, we have a special category, the test drive car, which has a much higher selling price than

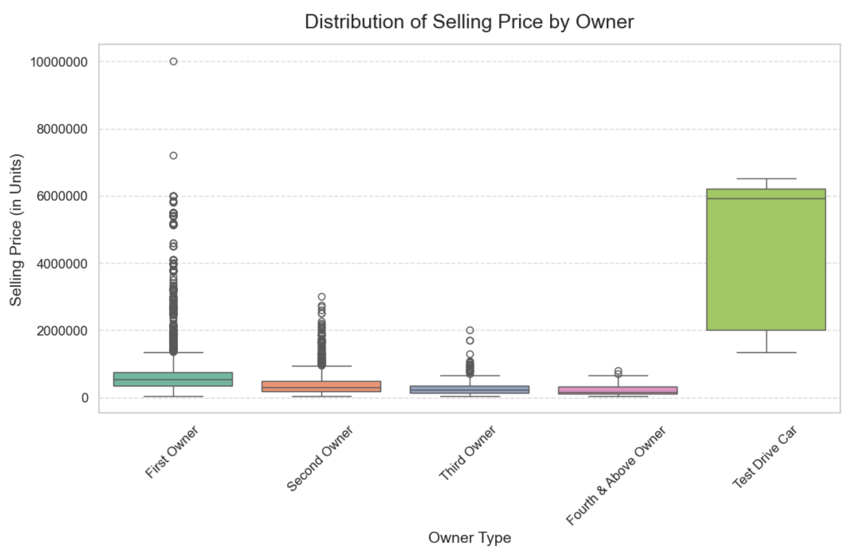
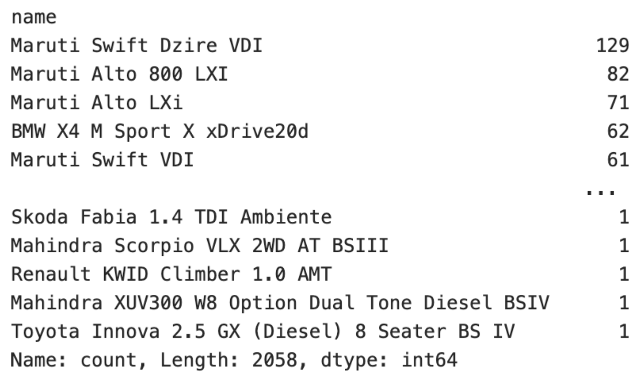


Fig 4. Bar plot of Selling Price by Owner

all other categories. We may further consider using ordinal encoding for this feature. Since it is less used than “First Owner Car”, we may consider encoding it as a 0 in the ordinal categories.

Furthermore, I examined the unique entries in the "name" column, shown in Fig. 5, and identified 2,058 distinct names. Drawing on insights from previous work, such as Igor Vladimirovich Lapshin's Car Price Prediction Analysis, I chose not to discard this feature. Instead, I extracted the car brand (the first word of the name) to evaluate its potential contribution to the predictive model.



```
name
Maruti Swift Dzire VDI                                129
Maruti Alto 800 LXI                                    82
Maruti Alto LXi                                        71
BMW X4 M Sport X xDrive20d                             62
Maruti Swift VDI                                       61
                                                     ...
Skoda Fabia 1.4 TDI Ambiente                            1
Mahindra Scorpio VLX 2WD AT BSIII                       1
Renault KWID Climber 1.0 AMT                            1
Mahindra XUV300 W8 Option Dual Tone Diesel BSIV         1
Toyota Innova 2.5 GX (Diesel) 8 Seater BS IV            1
Name: count, Length: 2058, dtype: int64
```

Fig 5. Value counts of name

3 Methods

3.1 Splitting Strategy

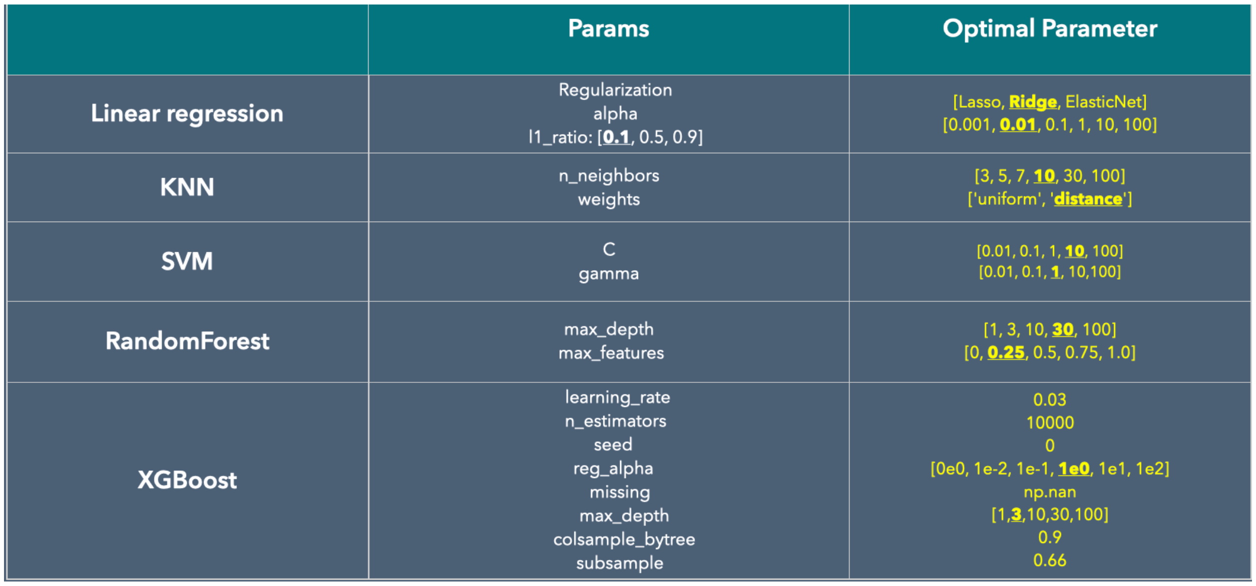
I first used the train-test-split to split 20% of the dataset into the test set. Then for the rest, 80%, I used KFold validation to ensure that every observation from the original dataset has a chance of appearing in the training and test sets, thus providing a thorough mixing of the data.

3.2 Preprocessing

I treated the log-transformed selling price as my target variable. For the remaining features, I used the ColumnTransformer to deal with different types of features. For categorical features fuel, seller_type, transmission, and brand, I did the one-hot encoding using `OneHotEncoder` with `handle_unknown='ignore'` to account for unseen categories. For ordinal feature owner, I transformed using an OrdinalEncoder with a predefined category order: ['Test Drive Car', 'First Owner', 'Second Owner', 'Third Owner', 'Fourth & Above Owner']. For continuous features year, km_driven, and engine, I standardized using StandardScalar since they don't have a scalable range, and for the ones that have a set range, which are mileage(km/ltr/kg), max_power, seats, I used MinMaxScalar to scale them. Moreover, since there are four columns with missing values, and the missing portion is approximately 2.7%, I imputed them using IterativeImputer, which captures the underlying relationships between features and reduces bias and variance introduced by missing data. After preprocessing, my number of features increased from 11 to 48.

3.3 Hyperparameters Tuning

I applied five ML algorithms, including both linear and non-linear models: Linear Regression (Lasso, Ridge, ElasticNet), K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Random Forest, and XGBoost. For each model, hyperparameter tuning was conducted using grid search.



	Params	Optimal Parameter
Linear regression	Regularization alpha l1_ratio: [**0.1**, 0.5, 0.9]	[Lasso, **Ridge**, ElasticNet] [0.001, **0.01**, 0.1, 1, 10, 100]
KNN	n_neighbors weights	[3, 5, 7, **10**, 30, 100] ['uniform', '**distance**']
SVM	C gamma	[0.01, 0.1, 1, **10**, 100] [0.01, 0.1, **1**, 10,100]
RandomForest	max_depth max_features	[1, 3, 10, **30**, 100] [0, **0.25**, 0.5, 0.75, 1.0]
XGBoost	learning_rate n_estimators seed reg_alpha missing max_depth colsample_bytree subsample	0.03 10000 0 [0e0, 1e-2, 1e-1, **1e0**, 1e1, 1e2] np.nan [1,**3**,10,30,100] 0.9 0.66

Table 1. Summary of Hyperparameters Tuning for each ML algorithm

The table above (Table 1.) summarizes the hyperparameter tuning process for various machine learning models to optimize their performance, and the bolded item has the best performance. For Linear Regression, regularization parameters such as alpha and l1_ratio were tuned, with Ridge regression emerging as optimal with an alpha value of 0.01. For KNN, the number of neighbors (n_neighbors) and weighting strategy (weights) were explored, with 10 neighbors and distance weights providing the best performance. In SVM, parameters C (regularization) and gamma (kernel coefficient) were tuned, with optimal values identified as C=10 and gamma=1. Random Forest hyperparameters included max_depth and max_features, with a max_depth of 30 and max_features=0.25 yielding the best results. Lastly, for XGBoost, a comprehensive grid search was conducted over multiple parameters such as learning_rate, n_estimators, max_depth, and regularization terms (reg_alpha). The best combination included a learning_rate of 0.03, max_depth=10, and reg_alpha=1e0, among others. These fine-tuned parameters were selected to maximize model accuracy while balancing overfitting and computational efficiency. A 4-fold cross-validation (`cv=4`) was employed to evaluate the model performance using the RMSE (Root Mean Squared Error) as the scoring metric.

3.4 Model Evaluation

To measure uncertainties in the evaluation metrics, I repeated the entire process across five different random seeds and calculated the mean and standard deviations for RMSE. This accounted for variability due to data splitting and the inherent randomness of certain models, such as Random Forest and XGBoost. At each stage, considerations such as the impact of feature scaling, the nature of missing data, and the importance of hyperparameter optimization were carefully factored into the pipeline design. This comprehensive strategy ensured a fair comparison of models and robust performance evaluation.

4 Results

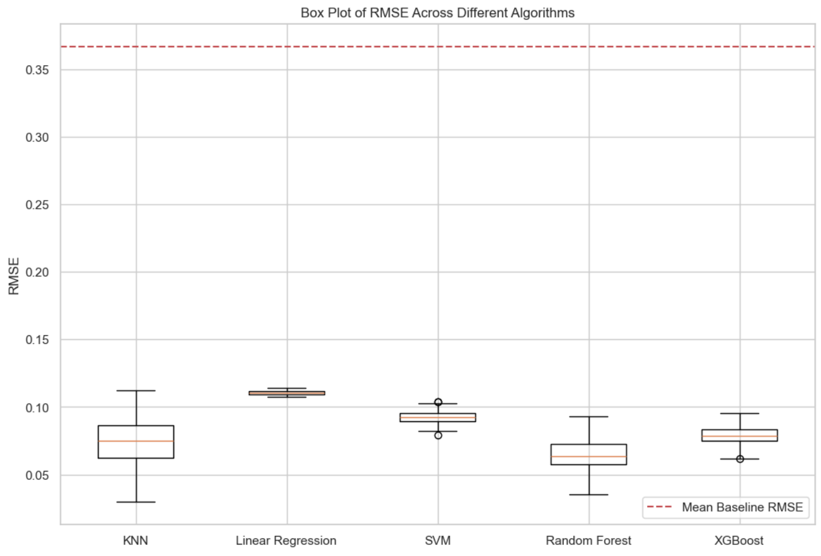


Fig 6. Box Plot of RMSE Across all Algorithms

$$\text{Baseline RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Fig 7. Formula of Baseline RMSE

Based on the graph (Fig 6.), the results demonstrate the effectiveness of various machine learning models in improving upon the baseline RMSE of 0.36694. The baseline RMSE was calculated using the formula shown in Fig 7. All models significantly outperformed the baseline, with Random Forest achieving the lowest mean RMSE of 0.067, approximately 17.64 standard deviations below the baseline. KNN also performed well, with a mean RMSE of 0.074 and a standard deviation of 0.017. XGBoost followed closely, with a mean RMSE of 0.079 and a standard deviation of 0.006. Among the tested models, Support Vector Machines (SVM) achieved a mean RMSE of 0.093, while Linear Regression had the highest RMSE of 0.111, though still a considerable improvement over the baseline.

Random Forest emerged as the most predictive model, exhibiting the lowest RMSE and a small standard deviation, indicating robust performance across different data splits. The box plot highlights the variability in model performance. These results underscore the value of ensemble methods like Random Forest and XGBoost for this dataset, given their ability to capture complex relationships and deliver consistent performance.

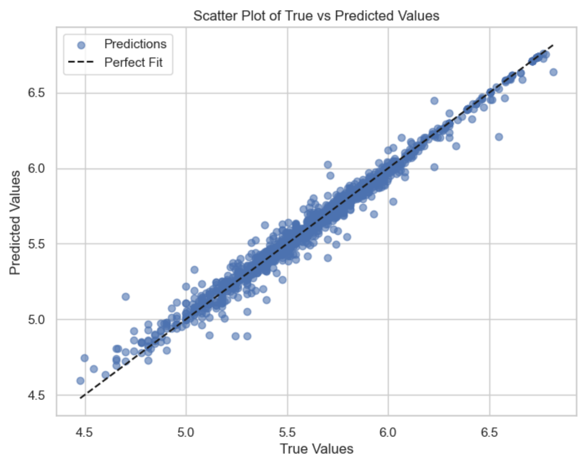


Fig 8. Scatter Plot of True vs. Predicted Values (Log Scale)

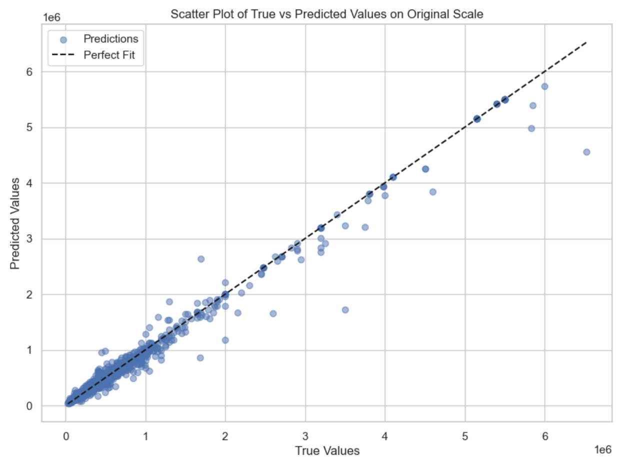


Fig 9. Scatter Plot of True vs. Predicted Values (Original Scale)

To further evaluate the model's predictive performance, I plotted the scatter plots of true vs. predicted values on both a logarithmic scale and the original scale. These visualizations provide key insights into the model's accuracy and error distribution.

In Fig 8, the model's predictions align closely with the diagonal "perfect fit" line. This indicates that the model performs exceptionally well in capturing the general trends in the data, especially when the logarithmic transformation reduces the impact of extreme values. Fig 9 allows for a more intuitive interpretation of the model's performance in the context of the raw data. While the majority of predictions remain close to the perfect fit line, a slight spread is observed, particularly for larger true values. The larger values may represent some luxury cars or some cars that are in a special edition.

In the real scenario, it means that error factor = 10^(0.067) = 0.17, which means that the predicted prices are, on average, within a factor of 1.17 of the true prices. In other words, the model's predictions are within ±17% of the actual prices. If the true price is 100000, the predictions will fall between 100000/1.17 and 10000*1.17.

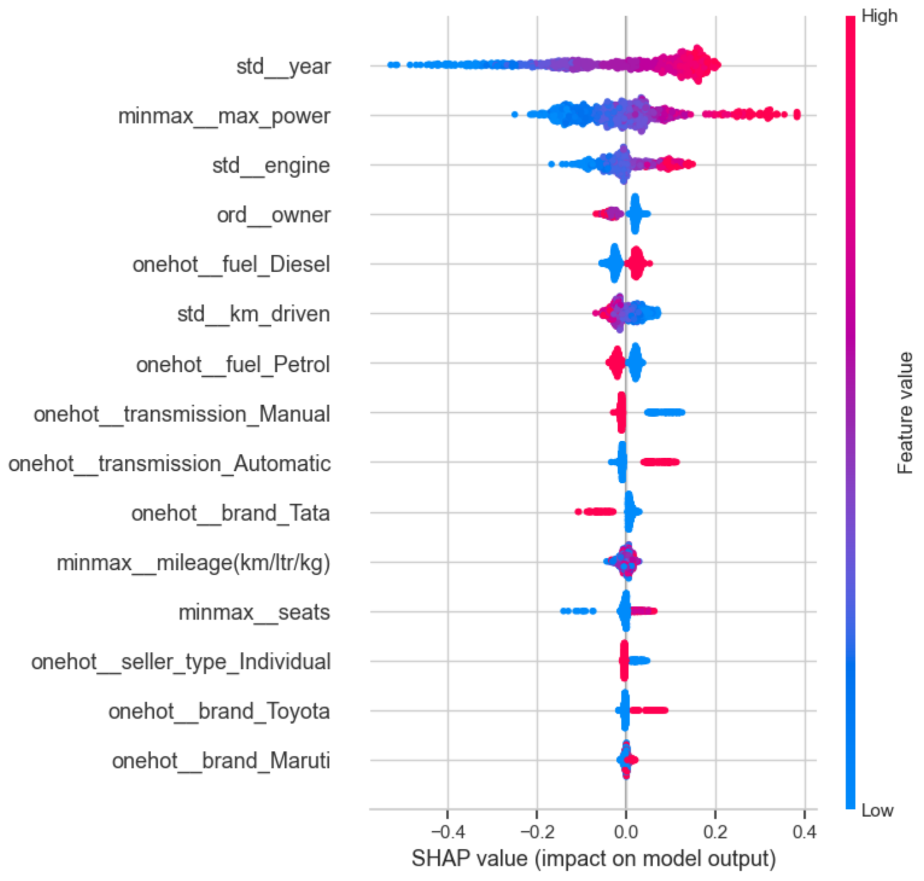


Fig 10. SHAP Summary Plot for Global Feature Importance

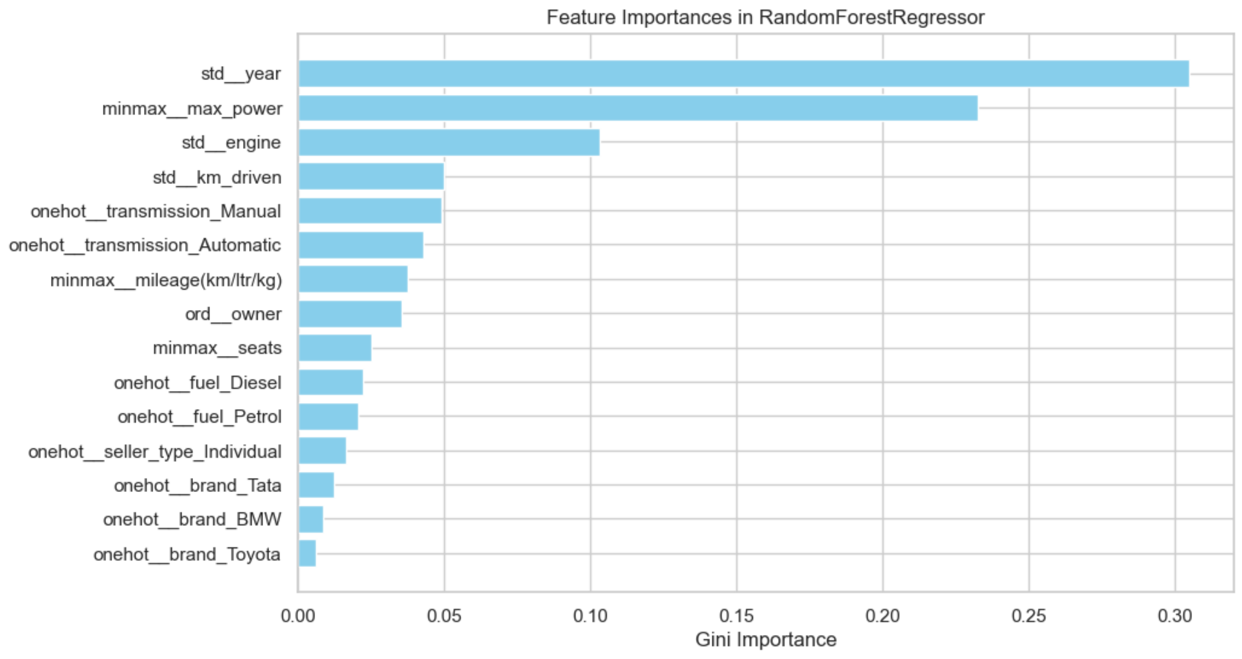


Fig 11. Gini Importance of Features in Random Forest Regressor

4.1 Feature Importance

The SHAP summary plot (in Fig 11) shows that year, max power, and engine size are the most influential features driving the model's predictions. These features make intuitive sense in the context of vehicle pricing, where newer cars, higher power, and larger engines often increase a car's value.

The Gini Importance plot (in Fig 12) shows the global feature importance derived from the Random Forest Regressor, where the importance of each feature is determined based on how much it reduces the model's overall error (impurity) during training. Features with higher Gini importance contribute more to the model's predictions.

The Permutation Importance plot shows the impact of each feature on the model's performance by measuring the increase in RMSE when the feature is randomly shuffled. Features that

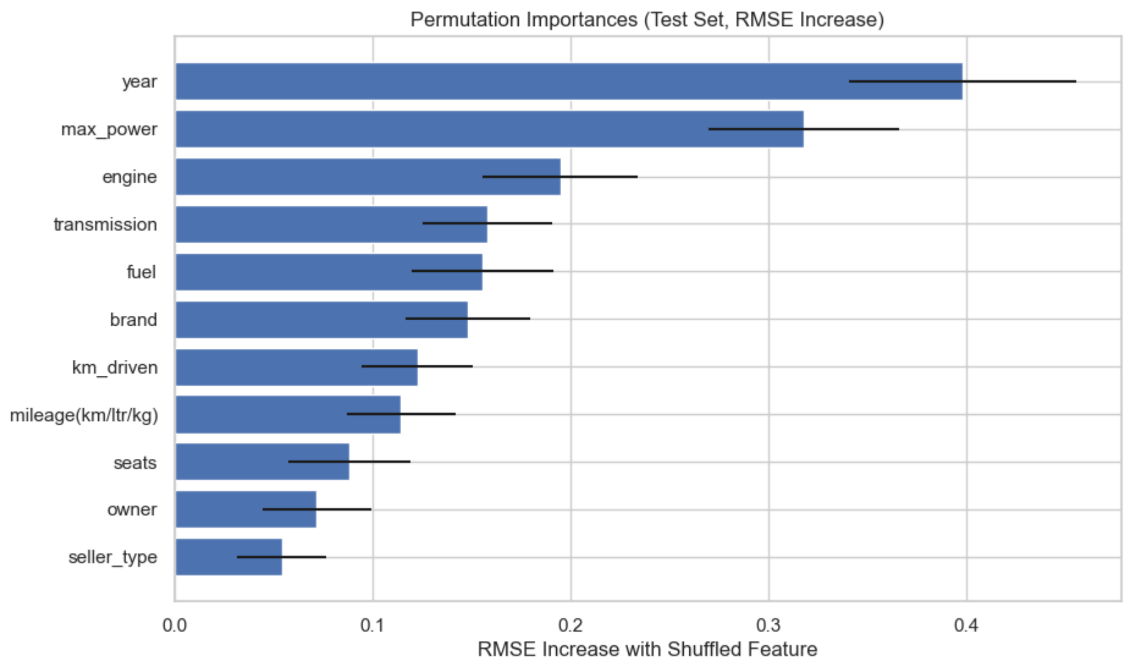


Fig 12. Permutation Importance plot

contribute significantly to the model's predictions will cause a larger increase in RMSE when disrupted, while less important features will have minimal impact.

All of those three plots consistently identify **"Year"**, **"Max Power"**, and **"Engine"** as the most critical features for predicting car prices. SHAP values provide insights into individual-level impacts, Gini importance highlights global model behavior during training, and permutation importance confirms the features' impact on unseen test data.

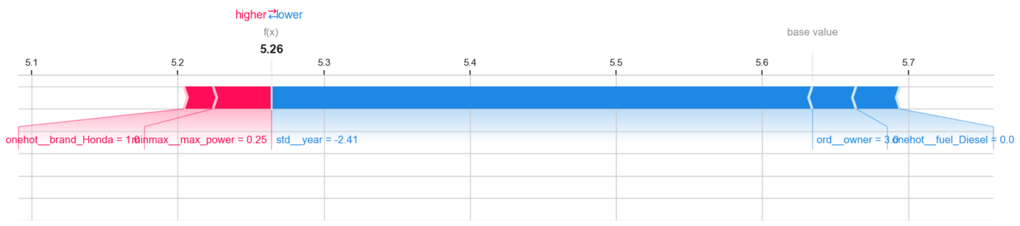


Fig 13. Local Feature Importance (SHAP Waterfall Plot) at data point 0

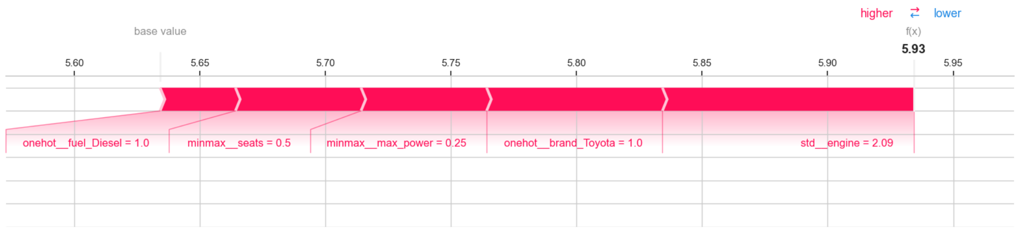


Fig 14. Local Feature Importance (SHAP Waterfall Plot) at data point 100

As shown above in Fig 13 and Fig 14, The SHAP waterfall plots highlight how individual features interact to drive model predictions for specific instances. Positive contributions, such as Diesel fuel, higher max power, and certain brands (Toyota or Honda), increase predictions, whereas features like older years and higher ownership act as strong negative influences. These local interpretations align with global feature importance results, confirming that the model effectively captures intuitive relationships, such as the positive value of newer, high-performance vehicles and the depreciation caused by age or multiple owners.

5 Outlook

To further enhance the performance and reliability of the model, instead of imputing all missing values, assess the importance of features with significant missing data. I will also try to expand the hyperparameter grid search or use advanced optimization techniques such as Random Search or Bayesian Optimization to explore a wider hyperparameter space. To improve interpretability, while SHAP values provide excellent global and local interpretability, tools like LIME can be explored to offer simpler, interpretable explanations for predictions on a case-by-case basis.

6 References

1. Car Price Prediction Analysis by Igor Vladimirovich Lapshin (Kaggle, 2024). https://www.kaggle.com/code/cardata/car-price-prediction-analysis
2. Kaggle Car prediction dataset (SUKHMANDEEP SINGH BRAR, 2024). https://www.kaggle.com/datasets/sukhmandeepsinghbrar/car-price-prediction-dataset/data
3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://scikit-learn.org/stable/