


Car Price Prediction Using Linear Regression and Random Forest Models

Gutter-Bacsi Zsombor

December 14, 2025

Abstract

This project explores used-car price prediction using a Kaggle dataset, combining data filtering and feature engineering with Linear Regression and Random Forest models. Random Forest achieved the best precision, while Linear Regression was sensitive to outliers. All code is available on  GitHub, and future work includes deploying a pricing API.

1 Introduction

For this project I decided to work with a real and somewhat messy dataset from Kaggle. After browsing several candidates, I chose the *Car Price Prediction Challenge* dataset [1]. I am not participating in the challenge itself, but I used its task description as the guiding idea of my work: build a model that predicts the price of a used car based on its attributes.

My goal was to achieve a model that reaches at least 80% practical prediction accuracy. Later I learned that depending on the chosen metric this number can be either meaningful or misleading. In other words, a model can be *statistically correct* yet still behave poorly for real cars.

2 Methods

My workflow evolved through four main stages: data handling, exploratory data analysis (EDA), the development of the string encoder, and finally the regression models. Although these steps are presented in a logical order, in practice I repeatedly circled back whenever I discovered a modelling issue. Many of the ideas that did not work were just as important as those that did, because they shaped the final design.

2.1 Data Handling

The project began by loading the raw Kaggle CSV into a custom `CarRecord` structure, which later produced a `Car` object containing an `Engine`. My intention was to keep all data in a neatly structured object-oriented format. In the long run this idea still has potential, because nested attributes (like engine data) can be handled naturally in objects.

However, once I started doing practical EDA, it became clear that `pandas DataFrame` was far more efficient. Most plotting, filtering, and statistical operations were dramatically easier. As a result, the `DataFrame` became the main working representation while the object model remained as a concept for future expansion.

2.2 Exploratory Data Analysis

I began EDA with simple statistics of prices and other numeric fields. The price distribution revealed an extremely wide range from practically zero to over 200 000 USD.

The correlation heatmap in Figure 1 revealed several important patterns. While some variables showed only weak direct correlations with price (for example mileage, around -0.01), others such as production year showed a moderate positive correlation (≈ 0.40). More importantly, using the *full* feature set exposed indirect relationships: production year is correlated with features such as leather interior, which themselves correlate with price. These multi-step dependencies would not be visible in a reduced heatmap containing only price and a few numeric variables. This can be seen in the pairwise correlation between price and production year, which appears weak in isolation, while the full heatmap reveals a clearer relationship once interactions with other features are taken into account.

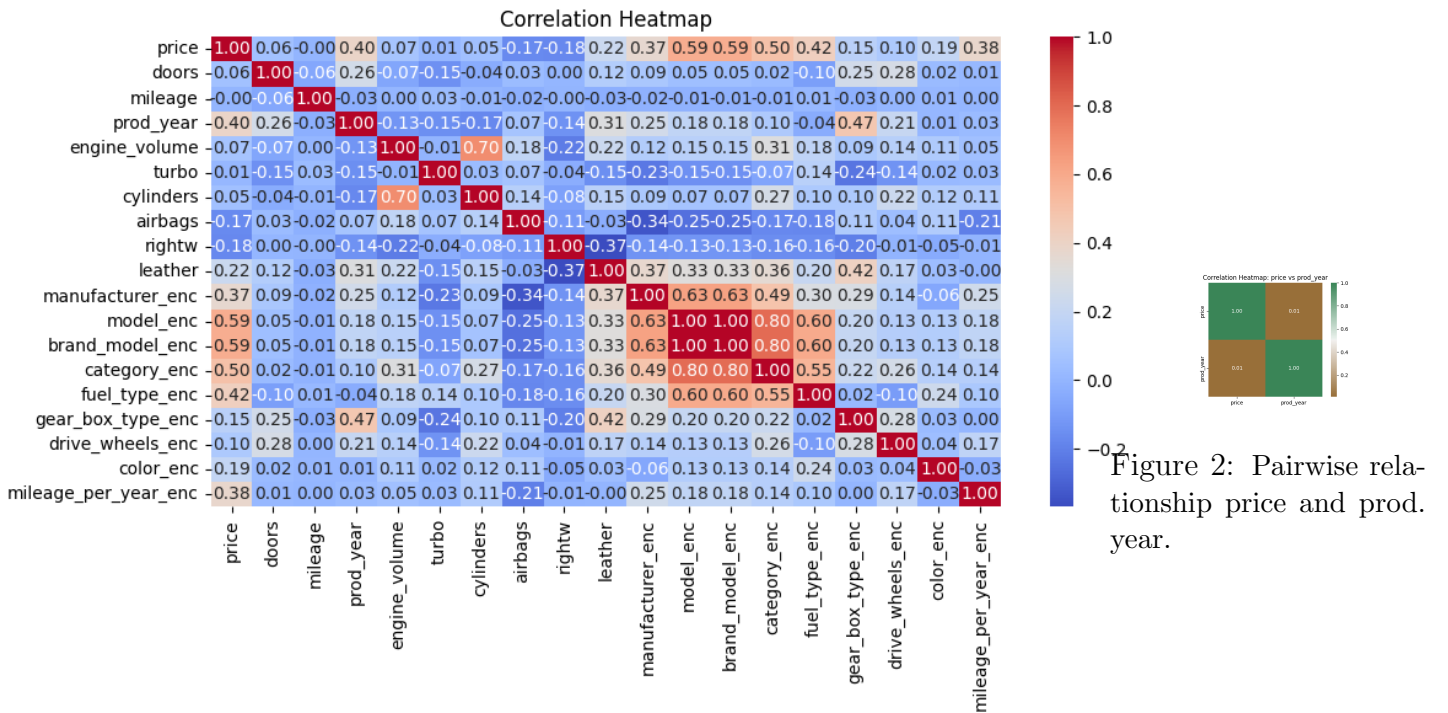


Figure 2: Pairwise relationship price and prod. year.

Figure 1: Correlation heatmap of all numerical and encoded categorical features.

Figures such as the price histogram and category distribution plots guided nearly all later filtering choices.

The manufacturer distribution in Figure 4 revealed a strong imbalance: many brands appeared only a handful of times. To stabilise the encoding process, all manufacturer and model categories with fewer than 180 entries were filtered out.

Another key discovery came from the distribution of prices shown in Figure 3. A surprisingly large number of cars were “dirt cheap” (often below 1000 USD), while a smaller but significant set was extremely expensive. These extremes do not behave like normal market prices and produced very large prediction errors. Removing low-end and high-end outliers based on $\ln(\text{price})$ thresholds (e.g., $\ln(p) < 7$ or $\ln(p) < 8$ and $\ln(p) > 12$) therefore proved crucial for stabilising both regression models.

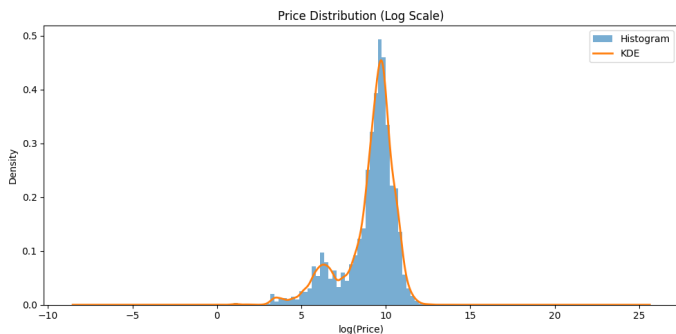


Figure 3: Price distribution histogram

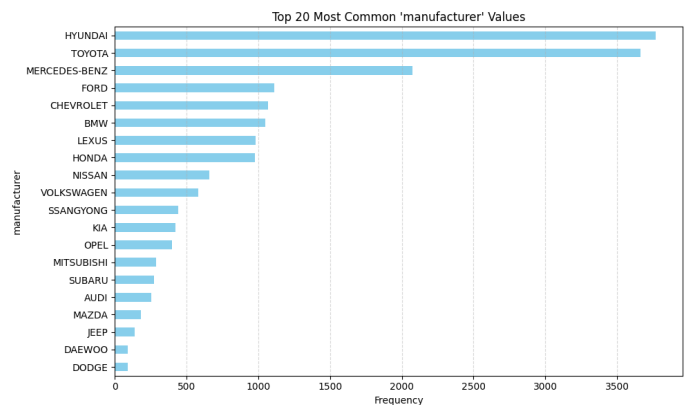


Figure 4: Manufacturer distribution

2.3 String Encoding and Feature Engineering

The encoder became the core component of the entire project. Both regression models depended heavily on the quality of these features.

(1) Mean-Price Encoding

A naive integer encoding of categorical variables (e.g., $\text{manufacturer} = \{0, 1, 2, \dots\}$) made no sense, because it would imply a ranking between brands. Instead, I assigned each category the *mean price* of cars in that category. For example, BMW naturally receives a higher encoded value than a budget brand if BMWs tend to be more expensive.

This worked extremely well for large categories like manufacturer or model, and reasonably well even for smaller categories such as fuel type or gearbox type. Although such smaller categories sometimes blur distinctions (high-end and low-end cars may share the same gearbox), in practice the encoding stabilised model behaviour more than expected.

(2) New Features

Two engineered features dramatically improved results:

- **brand_model**: a concatenation of manufacturer and model, allowing the model to distinguish within-brand price variation (e.g., BMW X5 vs. a cheaper BMW model).
- **car_age** and **mileage_per_year**: derived features capturing depreciation in a more informative manner than mileage or production year alone.

These combined features became some of the strongest predictors in both models.

(3) Filtering

Based on distribution plots and category counts, the encoder applies several filters:

- remove manufacturer and model categories with fewer than 180 occurrences,
- remove extremely cheap cars (likely sold for spare parts) and extremely expensive outliers,

- drop the `levy` field, which introduced noise without improving predictions (the tax affects the likelihood of purchasing a car, not its market price, and therefore does not belong in this model).

Filtering contributed more to the final performance than any model parameter tuning.

Failed and Abandoned Attempts

Several modelling ideas turned out to be misleading or ineffective:

- **Weighting encoded categories based on Random Forest feature importance.** Although theoretically appealing, neither Linear Regression nor Random Forest showed any meaningful change when weights were adjusted. Later I understood why: Linear Regression rescales coefficients automatically, and Random Forest is invariant to monotonic scaling of inputs.
- **Dimension reduction by dropping low-correlation features.** I attempted to remove all categories with $|r| < 0.25$ relative to price. The result was worse, because correlation alone does not capture non-linear or interaction effects which Random Forest benefits from.
- **Polynomial features.** Degree 2 gave a small improvement, but degree 3 and above exploded numerically. This is expected: polynomial features create severe multicollinearity unless carefully regularised.

These failed routes were useful because they clarified which components truly matter: clean data, well-designed encoders, and stable filtering rules.

3 Models

In this project I focused on two regression models, implemented using the *scikit-learn* machine learning library [2]:

- Linear Regression (simple, interpretable baseline)
- Random Forest Regression (non-linear, robust against noise)

(Both models will be evaluated in the 4.3 Model Comparison section.)

3.1 Linear Regression [3]

Linear regression is used as a baseline model due to its simplicity and interpretability. Conceptually, it can be seen as a direct generalization of linear approximation to higher dimensions. Instead of fitting a line to data, the model fits a hyperplane in a multi-dimensional feature space.

Given an input feature vector $\mathbf{x} \in \mathbb{R}^n$, the model predicts the target price \hat{y} as

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n,$$

where the coefficients β_i are learned by minimizing the mean squared error between predictions and true prices.

Linear regression assumes a linear relationship between the features and the target variable. While this assumption is clearly violated for many aspects of car pricing, the model still provides a useful reference point. Its coefficients allow direct inspection of how individual features influence the predicted price, which helps to validate preprocessing and encoding choices.

In practice, linear regression struggled to model complex interactions between features such as brand, model, and vehicle condition. This limitation motivated the use of a non-linear model.

3.2 Random Forest Regression [4]

Random Forest Regression is an ensemble-based, non-linear regression model built from a collection of decision trees. Each tree is trained on a random subset of the data and features, and the final prediction is obtained by averaging the predictions across all trees.

Unlike linear regression, random forests do not assume a linear relationship between inputs and the target variable. This allows the model to capture non-linear effects and interactions between features such as brand, mileage, production year, and engine characteristics, which are expected to be important for car price prediction.

While I am more familiar with linear regression and understand its behavior well, I encountered Random Forest Regression while inspecting related work by *nayan2112* [5]. Most other examples I found online relied primarily on linear regression models. Further investigation showed that random forests are well-suited for non-linear and heterogeneous data, which is why I included it.

Halfway through the project, I decided to work in parallel with both linear and random forest regression models. I generally treated the random forest model as the stronger candidate, but included it mainly for comparison, as I am less familiar with its internal behavior than with linear regression. Despite this, the linear model ultimately performed surprisingly well after sufficient data filtering and feature engineering.

*

More advanced models could potentially improve performance further, but the chosen models already demonstrate the strong influence of preprocessing, feature engineering, and data filtering on the prediction quality.

4 Results

The performance of the models was evaluated primarily using the Root Mean Squared Error (RMSE) and the mean and median of the absolute relative error. I also inspected the per-car percentage error to understand how the models behaved across different price ranges.

4.1 Evaluation Metrics

RMSE was my main metric. It is defined as

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

It heavily penalises large individual errors. This makes it useful for detecting cases where the model makes catastrophic mistakes, but also means that a single bad prediction can dominate the metric. In practice, I found that RMSE can be somewhat misleading: two models with different behaviours may have similar RMSE values, and a model with a high RMSE may still have a small average relative error.

Therefore, I additionally evaluated the models using per-car relative errors. For each car, I computed the absolute relative error

$$e_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100\%.$$

From this I calculate:

- **Mean absolute relative error** (average deviation across cars),
- **Median absolute relative error** (a central measure, less sensitive to outliers),
- **Relative precision**, defined as

$$\text{Precision} = 100\% - e.$$

Relative precision provides a more intuitive interpretation: it measures how close the prediction is to the true price in percentage terms. In several cases, a model with a worse RMSE still achieved higher relative precision, especially after applying logarithmic price filtering.

Runtime was not a bottleneck: both Linear Regression and Random Forest trained very quickly relative to dataset size, so time was not used as a comparison criterion.

4.2 Importance of Log-Based Price Filtering

A central observation was that extremely cheap cars (\$100–\$1000 range) and extremely expensive cars distort the regression behaviour. These vehicles do not behave like “normal market” prices, and they formed long tails in the distribution plots. Using the logarithm of price made it easy to identify meaningful thresholds. (see Figure 3).

- $\ln(p) > 7$ removes cars below roughly \$1100. This eliminates scrap-value or non-functional vehicles while still keeping a wide and realistic range of low, mid, and high-value cars.
- $\ln(p) > 8$ removes cars below roughly \$3000. The remaining dataset becomes more homogeneous and is dominated by mid- and high-value vehicles. In this regime, RMSE often increases because absolute price differences grow, but relative precision improves, indicating more consistent predictions within a narrower price spectrum.

This is an important result: filtering does not make the model “cheat”; it changes the problem definition to a more realistic pricing scenario.

4.3 Model Comparison

Table 1 summarises the performance of both models under the two filtering thresholds. Values are taken from independent runs.

Filter	Model	RMSE	Mean Rel. Preciseness	Median Rel. Preciseness
$\ln(p) > 7$	Linear Reg.	≈ 7900	25%	73%
	Random Forest	≈ 4650	73.68 %	90.08 %
$\ln(p) > 8$	Linear Reg.	$\approx 8100^1$	52.9%	75.06 %
	Random Forest	≈ 4550	79.48 %	90.62%

Table 1: Comparison of Linear Regression and Random Forest under two filtering thresholds. RMSE is in absolute price units. (\$)

4.4 Summary of Findings

The initial goal of this project was to achieve at least **80% prediction precision** using Linear Regression. This target was not consistently reached under any of the evaluated metrics. However, closer inspection shows that this result is strongly influenced by the error distribution rather than systematic model failure. While the **mean relative precision** was low, the **median relative precision** remained in the low-to-mid 70% range, indicating that most predictions were reasonably accurate.

The discrepancy between mean and median precision reveals a highly skewed error distribution: when Linear Regression makes an error, it is occasionally very large (sometimes exceeding 100%), but the majority of predictions fall within a small error range, often below 10% and in many cases below 5%. A small number of extreme outliers therefore dominate the mean metric and distort the overall evaluation.

¹RMSE increases for $\ln(p) > 8$ in Linear Regression because the remaining cars span a wider absolute price range, not because the model performs worse.

Random Forest Regression consistently outperformed Linear Regression in both RMSE and relative precision. Filtering with $\ln(p) > 7$ produced the best balance between dataset size and prediction stability, while $\ln(p) > 8$ further improved relative precision at the cost of higher RMSE, particularly for Linear Regression. The **best-performing** configuration was the **Random Forest model** with $\ln(p) > 7$, achieving RMSE values around 4600–5000 while maintaining high relative precision on the most intact dataset.

These findings suggest that additional pre-filtering or limited human oversight, such as removing severely under- or over-valued listings, could further improve Linear Regression performance and potentially bring it closer to the original precision target. Similar approaches were used in online examples such as Zabihullah’s notebook [6] and Nayan Shreemen Pale’s work [7]. Both achieved 80%+ precision with Linear Regression, but only by applying much heavier filtering than I considered acceptable. In Zabihullah’s case this was combined with more than a dozen additional datasets, many of which contained far richer and more detailed car attributes than the Kaggle [1] dataset I used. While effective for boosting accuracy, this level of filtering and dataset mixing in my opinion compromises the integrity of the original problem and narrows the prediction task to a much easier, more homogeneous subset of vehicles.

5 Future Work

Future work includes deploying the model as a car pricing API or web-based application. I started developing such application, but deployment will be postponed until model precision improves.

Further improvements could come from stricter data filtering, training separate models for low-, mid-, and high-value cars, and combining multiple datasets to improve generalisation. The existing object-oriented structure could also be extended for better scalability and reuse. Finally, more advanced regression models may be explored, although this project intentionally focused on Linear Regression and Random Forest methods.

References

- [1] Deep Contractor. Car price prediction challenge. <https://www.kaggle.com/datasets/deepcontractor/car-price-prediction-challenge>, 2022. [Online; accessed 12/14/2025].
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [3] Scikit Learn. Linearregression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html, 2025. [Online; accessed 12/14/2025].
- [4] Scikit Learn. Randomforestregressor. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>, 2025. [Online; accessed 12/14/2025].
- [5] nayan2112. Car-price-prediction. <https://github.com/nayan2112/Car-Price-Prediction>, 2021. [Online; accessed 12/14/2025].
- [6] Zabihullah1. Car price prediction. <https://www.kaggle.com/code/zabihullah18/car-price-prediction/notebook>, 2024. [Online; accessed 12/14/2025].
- [7] Nayanshree Menpale. linearregression_carpriceprediction. <https://www.kaggle.com/code/nayanshreemenpale/linearregression-carpriceprediction/notebook>, 2025. [Online; accessed 12/14/2025].