Abhi Reddy

Predicting Used Car Prices with Classical Machine Learning

Introduction and Project Statement

Buying or selling a used car is something a lot of people do, but figuring out a fair price is harder than it looks. Two cars that seem similar can differ in price because of mileage, age, fuel type, ownership history, transmission, seller type, and brand effects. The goal of this project was to build a model that can predict a used car's selling price from basic listing information, and then use the model results to better understand what factors drive price in the data. I framed this as a supervised regression problem. Given a set of structured features about a used car listing, the model predicts a numeric selling price. Beyond prediction accuracy, I also wanted to compare simple models like linear regression and regularized linear models with non-linear models like tree ensembles, and to interpret the final model so the results aren't just a black box. This project is designed to be reproducible, with clear preprocessing steps, consistent evaluation metrics, and outputs that can be checked directly in the notebook. Each major step below maps to a section in the notebook, so it is easy to screen-record the report while showing the corresponding evidence and outputs in the notebook.

Data Sources and Technologies Used

I used the "Car Details from CarDekho" used-car listings dataset downloaded from Kaggle as a CSV file. The dataset contains 4,340 rows and 8 columns in its original form. Each row is one car listing with fields including name, year, selling price, km driven, fuel, seller type, transmission, and owner. The notebook shows the dataset shape immediately after upload and preview of the first few rows (Notebook cell 0). Feature engineering and cleaned dataset. After cleaning and feature engineering, the dataset remained 4,340 rows, but the feature set changed to 9 columns because I extracted a brand feature from the original name field and created car age from year while dropping the raw name string to avoid extremely high-cardinality categorical features. The notebook prints the After cleaning shape as (4340, 9) (Notebook cell 2). The main libraries were pandas and NumPy for data handling, scikit-learn for preprocessing, training, and evaluation, and matplotlib for plots. The final best model pipeline was saved using joblib. The project folder also includes a "Use of AI" log that tracks which code blocks were AI-generated and where they appear in the notebook.

Methods Employed

I started with a light EDA to make sure the dataset loaded correctly and to understand the

variables before modeling. In the notebook I inspected column names, summary statistics, missing values per column, and basic histograms for the key numeric features such as selling price, km driven, and year. This helped confirm that the target distribution is skewed, that there are a small number of high-price listings, and that mileage has a wide range. Even though the EDA was intentionally quick, it served two practical purposes: it prevented silent parsing errors and it guided the later decision to use models robust to skew and non-linearities.

Cleaning and Feature Engineering

The cleaning step focused on preventing obvious data quality issues from breaking model training or inflating evaluation errors. In the notebook, I converted selling price and km driven to numeric types, dropped rows missing the target, and removed invalid values such as non-positive prices or negative mileage. Because name contains useful information but is too high-cardinality to one-hot encode directly, I extracted a simpler brand feature, first token of the original name string, and dropped the raw name column. I also created car age from year using the current runtime year. This produced a clean, model-ready dataframe with a mix of numeric and categorical features.

Train/Validation/Test Splitting

To evaluate models fairly, I used a three-way split. The notebook first holds out a test set, then splits the remaining data into train and validation. The final split sizes were Train: 2,951 rows, Validation: 738 rows, and Test: 651 rows (Notebook cell 3). The validation set was used for model selection and light hyperparameter tuning, and the test set was only used once at the end for the best model to avoid optimistic bias

Preprocessing Pipeline

Because the dataset has both numeric and categorical variables, I used a scikit-learn ColumnTransformer preprocessing pipeline so that all models receive the same standardized feature representation. Numeric features (year, km driven, car age) were imputed with the median and then standardized with StandardScaler. Categorical features (fuel, seller type, transmission, owner, brand) were imputed with the most frequent value and one-hot encoded with OneHotEncoder to prevent errors when a category appears in validation or test but not train. The notebook prints the detected numeric and categorical feature lists, which is helpful evidence that the pipeline is using the intended columns (Notebook cell 4). This pipeline approach matters because it reduces hidden leakage and ensures reproducibility. If you train different models with different ad-hoc preprocessing, comparisons become unreliable. Using a single shared pipeline makes the comparison

cleaner and makes saving/loading the model straightforward because preprocessing is packaged with the estimator.

Models Trained

I trained and compared six models, all using the same preprocessing pipeline:

1. Dummy baseline (median predictor). This is a sanity-check baseline that predicts the median training price for every car. It sets a floor for how well a model must perform to be meaningful.
2. Linear Regression. A standard linear model to establish an interpretable baseline.
3. Ridge Regression. A regularized linear model that can improve generalization in the presence of correlated features and many one-hot categories. I tuned the regularization strength alpha using 5-fold cross-validation over a log-spaced grid.
4. Lasso Regression. Another regularized linear model that can drive some coefficients to zero (feature selection behavior). I tuned alpha similarly. During training, scikit-learn raised a convergence warning, which is common for Lasso on larger sparse feature spaces; the model still produced usable results, but it also indicated that Lasso may not be the best fit for this dataset without further tuning.
5. Random Forest Regressor. A non-linear tree ensemble designed to capture interactions between variables. For example, the effect of mileage can depend on the car's age and brand. I used 400 trees with a fixed random seed.
6. Gradient Boosting Regressor. Another tree ensemble that often performs strongly on tabular data, included as a competitive non-linear comparison.

All models were evaluated with the same metrics: MAE, RMSE, and R^2. MAE is especially useful here because it measures average error in the same units as the target price, which is easier to interpret than squared error alone.

Results

Validation Comparison (Model Selection)

Model selection was done using validation MAE.. The notebook prints a full results table with train and validation metrics for each model (Notebook cell 5). The table below reproduces those outputs:

Model	Train MAE	Train RMSE	Train R^2	Val MAE	Val RMSE	Val R^2
RandomForest (n=400)	53,328.03	107,967.90	0.9693	111,816.82	203,013.77	0.8329

GradientBoostingRegressor	130,997.29	201,838.36	0.8928	137,394.57	208,309.34	0.8241
LinearRegression	181,667.46	337,578.96	0.7002	175,779.54	294,815.51	0.6476
Ridge (best $\alpha = 0.01$)	181,690.41	337,579.61	0.7002	175,783.29	294,795.61	0.6477
Lasso (best $\alpha = 0.001$)	181,666.01	337,578.95	0.7002	176,967.61	297,574.91	0.6410
DummyRegressor (median)	311,946.49	635,949.36	-0.0638	285,855.02	511,669.91	-0.0614

This comparison shows a clear pattern. The median baseline performs poorly (negative R^2) as expected. Linear models Linear/Ridge/Lasso improve substantially, reaching validation R^2 around 0.65, but they plateau at a relatively high MAE. Tree ensembles outperform linear models by a wide margin, with Random Forest achieving the best validation MAE and a much higher R^2. The improvement strongly suggests that the relationship between price and features is not purely linear; interactions like "mileage matters differently depending on age and brand" are likely important. A noteworthy detail is the training vs validation gap for Random Forest. Training MAE is much lower than validation MAE, which indicates some overfitting, but the validation performance is still strong and generalizes well on the final test set. In contrast, linear models have a smaller train/validation gap but a weaker overall fit, implying underfitting relative to the data complexity.

Final Test Performance

After selecting Random Forest as the best model on validation MAE, I refit it on the combined train+validation set and evaluated once on the held-out test set (Notebook cell 6). The final test metrics were:

- Test MAE: 104,255.22
- Test RMSE: 171,943.40
- Test R^2: 0.8692

These results indicate that, on average, the model's predicted selling price is about 104k units away from the true value, and the model explains roughly 87% of the variance in the test-set prices. For a real-world marketplace dataset with limited features and a skewed price distribution, this is a strong outcome and shows that classical ML methods can be effective for pricing estimation even without text descriptions, photos, or detailed mechanical history.

Diagnostic Plots and Error Behavior

The notebook includes a Predicted vs Actual scatter plot and a residual histogram (Notebook cell 7). The Predicted vs actual plot shows most points clustered along the diagonal trend for the common price range, which indicates the model is generally tracking the target well. The spread increases for higher-priced cars, which is expected because there are fewer expensive listings and they may differ in ways not captured by the available features like trim level, modifications, condition, accident history, etc. The residual histogram is centered near zero, which suggests the model does not have a large overall bias toward overprediction or underprediction, but it has long tails. Those tails represent the cases where the model is most wrong, and they likely correspond to unusual listings or outliers.

Interpretation/What the Model Learned

The notebook also generates a feature importance plot for the best model (Notebook cell 8. While exact importances depend on the trained forest, the most influential factors typically include car age/year and kilometers driven, followed by categorical differences such as brand, fuel type, and transmission. This aligns with intuition. Older cars and higher-mileage cars tend to sell for less, and the same mileage/age can carry different implications depending on brand and fuel type. The interpretation step is important because it turns the model from “just a predictor” into something that can be explained and defended.

Practical Output

Finally, the notebook saves the best model as a single reusable pipeline, the best used car price model.joblib, that includes both preprocessing and the estimator. This is useful because it means new data can be passed in the same raw column format and the pipeline will automatically apply the same transformations before predicting.

References

CarDekho dataset: "Car Details from CarDekho" / "Vehicle dataset from Cardekho"
https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho?resource=download

scikit-learn documentation (models, preprocessing, and metrics)
https://scikit-learn.org/stable/

joblib (model persistence used by scikit-learn)
https://joblib.readthedocs.io/

Course learnings
https://coe379l-fa25.readthedocs.io/en/latest/index.html