Predicting Used Car Prices Using Machine Learning

1. Problem Statement and Motivation

The process of buying and selling used cars is often opaque: prices vary by location, trim, mileage, and listing quality. Consumers and private sellers lack a single, data-driven source of truth to estimate a fair market price for a specific vehicle. This leads to negotiation inefficiencies, buyer regret, and time wasted comparing listings manually.

This project builds a predictive model for used car prices using a large scraped dataset. The primary aim is to produce a model that can estimate a realistic price given commonly available listing attributes (year, mileage, model, engine description, etc.). The model is intended as a decision-support tool for buyers, sellers, and small dealerships.

Motivating example: Alice wants a 2018 Toyota Camry with 35,000 miles. Listings vary widely; a model trained on many past transactions can produce a single recommended price (and a confidence estimate), streamlining negotiation.

2. Data

We used the **"Used Cars Dataset"** (scraped from cars.com, available via Kaggle). The cleaned dataset for this project contains 762091 rows after filtering and sanitization. Each listing contains many fields; the most important ones for pricing are shown below.

2.1 Dataset Fields (selected)

Feature	Type	Description
manufacturer	categorical	Vehicle brand
model	categorical	Model name / trim
year	numeric	Year of manufacture
mileage	numeric	Odometer reading in miles
engine	text	Engine description; parsed into subfeatures
transmission	categorical	Automatic, manual, CVT, etc.
drivetrain	categorical	FWD, RWD, AWD/4WD
fuel_type	categorical	Gasoline, hybrid, electric, diesel
price	numeric	Listing price (target)

2.2 Filtering and initial cleaning

The raw dataset contains noise (currency symbols, text artifacts, missing values). The following filters and cleaning steps were applied:

- Price converted to numeric and rows with price < $2,000 or > $120,000 removed.
- Year restricted to 1995–2024 to avoid antique / misreported years.
- Mileage constrained to 0–300,000 miles.
- Non-numeric characters removed from numeric columns; missing numerics imputed later during preprocessing.

2.3 Descriptive statistics (sample)

Feature	Mean	Median	Min	Max
Year	2016	2017	1980	2025
Mileage	52,000	45,000	0	300,000
Price	$16,500	$15,000	$2,000	$150,000

Note: summary statistics computed after the filtering step described above. Price distribution is right-skewed, motivating a log-transform on the target for modelling stability.

3. Exploratory Data Analysis (EDA)

EDA focused on relationships known to affect price: age, mileage, manufacturer and model, and engine type.

3.1 Age & Mileage

As expected, price decreases with age and mileage. Age (computed as 2025 - year) and mileage per year are strong predictors. Many listings have low-mileage recent cars; these concentrate in a higher-priced mode.

3.2 Manufacturer & Model effects

Premium brands (e.g., BMW, Mercedes-Benz) command higher average prices. High-cardinality categorical variables such as model and manufacturer were reduced to the top-N categories and the remainder labeled 'Other' to control dimensionality for one-hot encoding.

3.3 Engine parsing

The free-text engine field was parsed to extract:

- displacement (liters),
- number of cylinders,
- turbo / supercharger flag,
- hybrid & electric flags.

These parsed features provide additional predictive signal: hybrids and EVs often carry a price premium for comparable age and mileage.

4. Data Robustness & Preprocessing

- Missing numeric features imputed with medians; missing categoricals filled with 'Unknown'.
- Outliers in price and mileage clipped/filtered as described in section 2.2.
- Top categories preserved for manufacturer/model (TOP_N constants used) and the rest grouped as 'Other'.
- Target price log1p-transformed to reduce heteroscedasticity.

5. Methods

5.1 Feature engineering

- **Age** = 2025 - year
- **Miles per year** = mileage / max(age, 1)
- **Engine parsing** → engine_disp_l, engine_cyl, engine_turbo_or_super, engine_is_hybrid, engine_is_electric
- **Cardinality reduction** for categorical fields (model, manufacturer, colors)
- **Scaling & encoding:** StandardScaler for numerics; OneHotEncoder for categorical variables

5.2 Model selection rationale

A linear baseline (Ridge) was evaluated but performed poorly ($R^2 < 0$), showing that relationships are nonlinear and interaction-rich. Gradient-boosted decision-tree models capture non-linearities and interactions between age, mileage, and categorical features and are robust to feature scaling. For this project we used scikit-learn's `HistGradientBoostingRegressor` for its performance and native handling of missing values.

5.3 Training setup

Data split: 80% training / 20% testing.

Target transformed: `y_log = log1p(price)`.

Key HGBR hyperparameters used (final model): `max_iter=300`, `learning_rate=0.05`, `max_depth=12`, `early_stopping=True`, `validation_fraction=0.1`, `random_state=42`.

5.4 Preprocessor (summary)

- Numeric pipeline: median imputation + StandardScaler
- Categorical pipeline: constant imputation ('Unknown') + OneHotEncoder
- Selected numeric features (example): ['age','mileage','engine_disp_l','engine_cyl']
- Selected categorical features (example): ['manufacturer','model_reduced','transmission']

5.5 Implementation notes

The preprocessor is built with `ColumnTransformer`. OneHotEncoder is used with `handle_unknown='ignore'` to allow safe inference on new data. Preprocessor is fitted on training data and persisted with `joblib`.

6. Results

Primary performance (reported on original price scale):

Model	RMSE	MAE	R^2
HGBR	$5,819	$3,655	0.888

6.1 Interpretation of metrics

An RMSE of ~$5.8k on a mean price of ~$16.5k indicates reasonable predictive ability; MAE of ~$3.6k means the median absolute pricing error is a few thousand dollars — useful for framing negotiation ranges but not a perfect replacement for localized appraisal. R^2 = 0.888 indicates the model explains a substantial portion of variance in price on the test set.

6.2 Feature importance & model behavior

Although tree-based models do not provide coefficients like linear models, feature importance (by permutation or built-in importance measures) highlights:

- **Age** and **mileage** — dominant numeric predictors
- **Manufacturer / Model** — large categorical effect, especially premium vs economy brands
- **Engine flags** — hybrid / electric / turbo provide modest adjustments

For transparency, the project recommends using SHAP values and partial dependence plots in future work to show per-listing contributions to predicted price.

6.3 Mini Case Example

Car	Age	Mileage	Manufacturer	Model	Predicted Price
Toyota Camry	3 years	30,000	Toyota	Camry	$17,200

6.4 Error analysis (qualitative)

Examining high-error predictions shows common failure modes:

- Local market differences — some markets pay premiums (e.g., EV demand) not captured without geolocation.
- Missing condition or accident data — listings missing condition metadata cause larger errors.
- Rare models / trims — low-sample models are hard to predict despite grouping into 'Other'.

7. Hyperparameter tuning & ablation

Rather than an exhaustive grid search on the entire dataset (computationally expensive), a pragmatic approach was taken:

- Conduct coarse grid search on a representative subset (`nrows` arg can be used to run smaller experiments).
- Key parameters tuned: `max_iter` , `learning_rate` , `max_depth` .
- Final values chosen balanced training time and validation performance (`max_iter=300` , `lr=0.05` , `max_depth=12`).

Ablation experiments (brief): removing engine-parsed features slightly reduced performance; collapsing model categories more aggressively reduced overfitting but cost some predictive power for niche trims.

8. Deployment considerations

To deploy this model in production, consider:

- Expose a small API that accepts listing attributes (manufacturer, model, year, mileage, engine text) and returns a predicted price + uncertainty range.
- Persist both `preprocessor.joblib` and `model_hgb.joblib` for inference.
- Monitor drift: auto-evaluate incoming predicted vs. realized sale prices and retrain periodically (monthly or quarterly).
- Geographic features: collecting listing ZIP/postal codes improves local pricing and should be included in the pipeline.

9. Limitations

- Missing condition and accident history in many listings limits accuracy.
- Regional price differences not captured without location features.
- Dataset bias: popular brands and regions overrepresented, so performance may vary on underrepresented markets.

- Extremely rare models or misreported fields can produce poor predictions.

10. Reproducibility & runtime

Reproducibility steps:

1. Run the pipeline with the same `--nrows` and `--topn_models` parameters for identical preprocessor columns.
2. Fix `random_state=42` to ensure deterministic train-test splits.
3. Persist artifacts with `joblib.dump()` and document package versions (scikit-learn >= 1.0 recommended).

Hardware note: training time depends on CPU cores and dataset size; HistGradientBoostingRegressor benefits from multiple threads but will still take longer on millions of rows.

11. Ethical considerations

Using data-driven pricing tools can influence markets. Common considerations:

- **Fairness:** If the training data reflects price discrimination (e.g., region-based premiums tied to demographic covariates), models can perpetuate inequities. Avoid including protected attributes.
- **Transparency:** Provide explanations for predictions (feature importance or SHAP) so users understand recommended prices.
- **Privacy:** Ensure scraped data usage complies with terms of service and privacy regulations.

12. Conclusion & Future Work

HGBR achieved strong predictive performance ($R^2 \approx 0.888$) and is a promising basis for a buyer/seller price recommendation tool. Future work should focus on conditioning on location, integrating vehicle condition and accident history, enabling model explainability with SHAP, and building a lightweight web API or dashboard for user interaction.

13. Appendix A — Pipeline summary & usage

Command-line usage summary for the training script (example):

```
# Train on the full dataset and save artifacts to results/
python used_car_price_project_hgbr_only.py --data cars.csv --outdir results --mode train

# Dry-run to inspect processed features (no training)
python used_car_price_project_hgbr_only.py --data cars.csv --mode dryrun --nrows 10000
```

13.1 Files produced

- `results/model_hgb.joblib` — trained HistGradientBoostingRegressor
- `results/preprocessor.joblib` — fitted ColumnTransformer
- `results/feature_columns.json` — list of final input columns used
- `results/metrics_summary.json` — saved RMSE/MAE/R^2 and row counts

14. Appendix B — Key functions (excerpt)

Below is a short excerpt of the key helper used to parse engine text (full script saved separately in code repository).

```
def parse_engine_text(s: str):
    if not isinstance(s, str):
        return (np.nan, np.nan, 0, 0, 0)
    text = s.lower()
    # displacement
    disp = None
    m = _engine_disp_re.search(text)
    if m:
        try:
            disp = float(m.group(1))
        except:
            disp = None
    # cylinders
    cyl = None
    m_cyl = re.search(r'(\d{1,2})\s*[-]?\s*(?:cyl|cylinder|cylinders)', text)
    if m_cyl:
        try:
            cyl = int(m_cyl.group(1))
        except:
            cyl = None
    # flags
    is_turbo = 1 if 'turbo' in text or 'turb' in text else 0
    is_super = 1 if 'supercharger' in text or 'supercharged' in text else 0
    is_hybrid = 1 if 'hybrid' in text else 0
    is_electric = 1 if ('electric' in text or ('ev' in text and 'engine' not in text)) else 0
    return (disp if disp is not None else np.nan,
            cyl if cyl is not None else np.nan,
            1 if (is_turbo or is_super) else 0,
            is_hybrid,
            is_electric)
```