# Car Resale Price Prediction using LightGBM

Project Report


Indian Institute of Information Technology, Design and Manufacturing, Kurnool

**Team:**

Aryan Rajesh Gadam (123cs0020)

Tagore Jagata (123cs0042)

Rovan Mullangi (123cs0036)

DOI: 10.1109/OTCON65728.2025.11070929

**Abstract**

The used car market in India is characterized by significant price volatility and information asymmetry, making it challenging for both buyers and sellers to determine fair vehicle value. This project addresses this problem by developing a robust machine learning model to predict the resale price of used cars. We utilize a comprehensive dataset of vehicle listings, applying rigorous data preprocessing, cleaning, and feature engineering to construct a predictive feature set. Key engineered features include vehicle age and an interaction term between age and kilometers driven.

We systematically compare the performance of three powerful ensemble models: Random Forest [4], XGBoost [3], and LightGBM [1]. The models are trained and evaluated using a pipeline that includes feature scaling and a log transformation of the target variable to handle its natural skewness. Our results demonstrate that the LightGBM model, optimized through `GridSearchCV`, achieves superior predictive performance, yielding a coefficient of determination ($R^2$) of 84.40% on the unseen test set. This report details the complete methodology, from dataset preparation and exploratory data analysis to model implementation, hyperparameter tuning, and comparative evaluation. Finally, we discuss the deployment of the optimized LightGBM model into a user-friendly Flask web application [8] for real-world inference.

## Keywords

Used Car Price Prediction, Machine Learning, LightGBM, Random Forest, XGBoost, Flask Web Application, Data Preprocessing, Feature Engineering, Regression, Python, India, Ensemble Methods.

# Contents

# List of Figures



# List of Tables

# 1 Introduction

The resale market for used automobiles is a significant and rapidly growing sector of the economy. Unlike new cars, which have standardized pricing, used car valuation is complex and depends on a multitude of factors, including brand reputation, model, age, mileage, condition, and market demand. This complexity often leads to information asymmetry, where sellers possess more information than buyers, creating market inefficiencies and a lack of trust.

Machine learning offers a powerful solution to this problem. By training models on historical sales data, we can uncover the intricate, non-linear relationships between a vehicle's attributes and its final selling price. The primary objective of this project is to develop and deploy a highly accurate predictive model for used car resale prices in the Indian market.

This report documents the end-to-end development of this predictive system. We begin with a description of the dataset and the exploratory analysis performed. We then detail the extensive data preprocessing and feature engineering steps, which are critical for model performance. The core of the project involves the comparative analysis of three state-of-the-art gradient boosting and ensemble methods: Random Forest [4], XGBoost [3], and LightGBM [1]. We evaluate these models rigorously, select the top performer (LightGBM), and analyze its behavior through feature importance and residual plots. Finally, we discuss the deployment of this model as a practical web application, providing a tool that can empower buyers and sellers with transparent, data-driven price estimates.

# 2 Dataset Description

## 2.1 Source and Overview

The primary dataset used for this project was `cardataset2.csv`. This raw dataset contained numerous inconsistencies, missing values, and unstructured text fields. After a thorough cleaning and preprocessing phase, we curated a final, model-ready dataset comprising 1,804 clean records. The dataset contains a rich mixture of numerical and categorical variables essential for car valuation.

## 2.2 Statistics

- Total initial records: 2,059
- Final clean records: 1,804
- Train/Test split: 80% train (1,443), 20% test (361)
- Price range: ₹141,000 to ₹35,000,000
- Mean price: ₹17,59,110

- Median price: ₹8,65,000
- Standard deviation: ₹24,56,231

## 2.3 Key Features

**Target:** `selling_price` (₹)

**Numeric features:** 7 features (Year, `km_driven`, `max_power_bhp`, `engine_cc`, `car_age`, `age_km_interaction`, `max_torque_nm`)

**Categorical:** 6 (including One-Hot Encoded and Ordinal)

## 2.4 Exploratory Data Analysis (EDA)

EDA was crucial for understanding feature distributions and their relationship with the target variable.

**Target Variable Distribution** The distribution of the `selling_price` (Figure 1) is heavily right-skewed. This indicates that most cars are clustered at the lower end of the price range, with a few luxury or high-performance vehicles as high-value outliers. This skewness justifies the use of a log transformation on the target variable before modeling to help normalize its distribution and stabilize variance, which is a common practice for regression models.

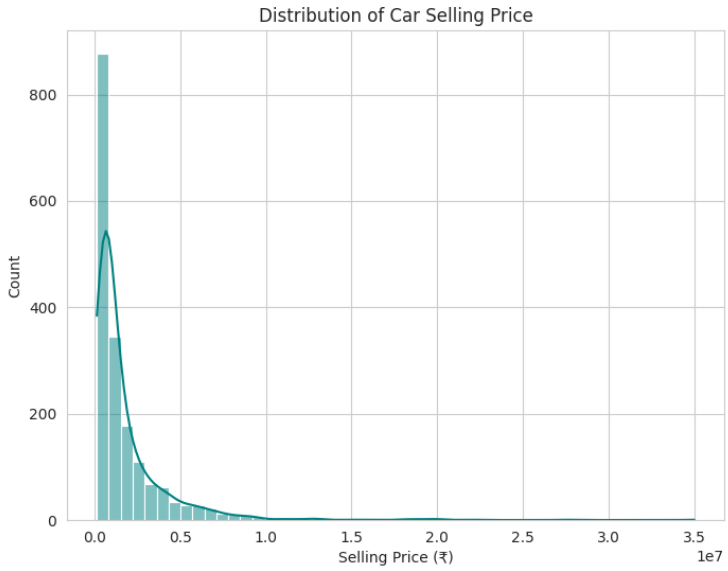



Figure 1: Distribution of Car Selling Price (₹), showing a strong right-skew.

**Feature Relationships** As shown in Figure **??**, strong positive correlations were observed between the selling price and key performance metrics. Both 'Max Power (bhp)' and 'Engine Size (cc)' show a clear trend: as these metrics increase, the selling price tends to increase as well. This aligns with the intuitive understanding that more powerful and larger-engine cars are generally more expensive. The plots also highlight the diversity of fuel types within different price and power brackets.

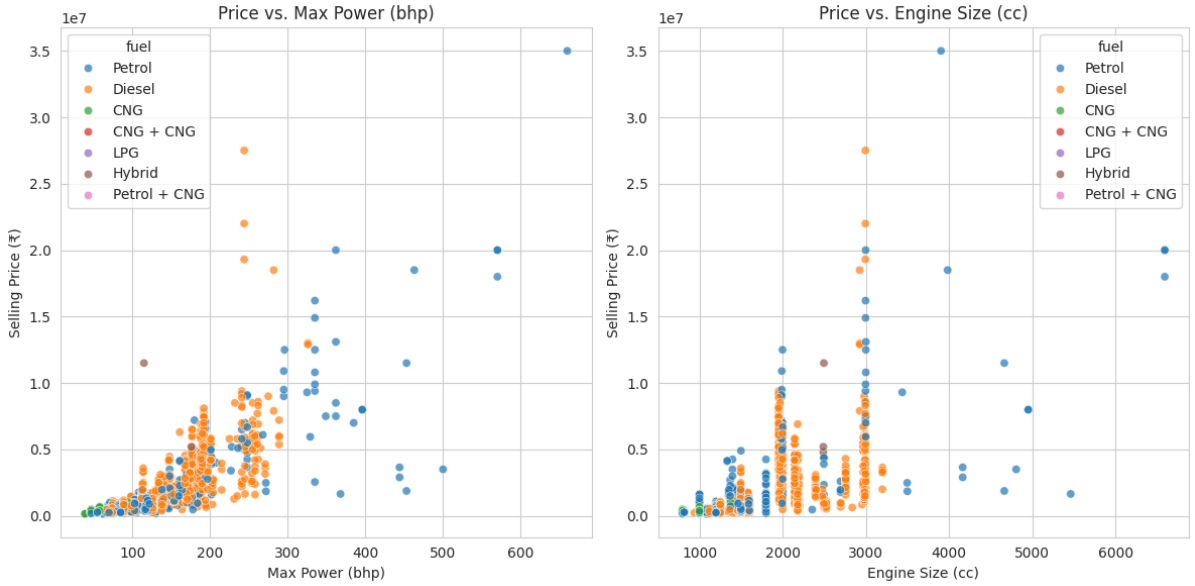



Figure 2: Relationship between selling price and key numeric features: Max Power (left) and Engine Size (right).

# 3 Data Preprocessing & Feature Engineering

## 3.1 Data Cleaning

Raw data requires significant cleaning to be useful for modeling. The following steps were performed:

1. **Numeric Extraction:** Extracted numeric values from unstructured text fields. For example, '120 bhp' was converted to '120.0' for `max_power_raw`, and '1998 cc' was converted to '1998.0' for `engine_raw`.

2. **Torque Standardization:** Processed the `max_torque` column, which had mixed units (Nm and kgm), to standardize all values into 'max_torque$_n m$'$(Newton - meters)$.

2. **Category Standardization:** Standardized owner categories (e.g., "First Owner", "Second Owner") to ensure consistency.

3. **Missing Value Imputation:** Handled missing values. For critical features like 'max$_p ower_b hp$', $recordsw$

### 3.2 Feature Engineering

New features were created to capture more complex relationships:

1. **car_age:** Calculated as '2025 - Year'. Vehicle age is often a more direct predictor of depreciation than the manufacturing year.
2. **age_km_interaction:** Calculated as 'car$_age$'× 'km_driven'. This interaction term captures the combined effect of age and usage. A new car with high mileage might depreciate differently than an old car with low mileage.
3. **max_torque_nm:** The standardized numeric torque feature.

### 3.3 Feature Encoding and Transformation

- **Numeric:** 7 features (Year, km_driven, max_power_bhp, engine_cc, car_age, age_km_interaction, max_torque_nm). These were scaled using `StandardScaler`.
- **One-Hot Encoded:** 4 low-cardinality categorical features (fuel, transmission, owner, Drivetrain).
- **Ordinal Encoded:** 2 high-cardinality categorical features (manufacturer, model_name).
- **Target Transformation:** The target variable `selling_price` was log-transformed ('np.log') to normalize its distribution, as identified during EDA. The model predicts the log of the price, which is then converted back using the exponential function ('np.exp') for evaluation.

## 4 Machine Learning Methodology

### 4.1 Model Pipeline Architecture

To ensure robust and reproducible preprocessing, we used Scikit-learn's [2] `Pipeline` and `ColumnTransformer`.

- **Preprocessing:** A `ColumnTransformer` applied `StandardScaler` to numeric features, `OneHotEncoder` to OHE features, and `OrdinalEncoder` to ordinal features.
- **Target Transformation:** We used `TransformedTargetRegressor` to wrap the entire process. This wrapper automatically applies the 'np.log' function to the target variable before training and the 'np.exp' inverse function to the predictions during inference.
- **Model:** The final step in the pipeline was the regressor model itself (e.g., `LGBMRegressor`).

### 4.2 Models Evaluated

We compared three powerful, tree-based ensemble models known for their high performance on tabular data:

1. **Random Forest Regressor [4]:** A bagging model that builds multiple decision trees on different subsets of data and averages their predictions to reduce variance.
2. **XGBoost Regressor [3]:** A gradient boosting model that builds trees sequentially, with each new tree correcting the errors of the previous ones.
3. **LightGBM Regressor [1]:** A more recent gradient boosting framework that uses a leaf-wise growth strategy, making it significantly faster and often more accurate than traditional level-wise boosting.

### 4.3 Hyperparameter Optimization

`GridSearchCV` with 3-fold cross-validation was employed to find the optimal hyperparameters for each model. The primary scoring metric for optimization and comparison was the $R^2$ (coefficient of determination).

### 4.4 Residuals Analysis

A residuals plot is a key diagnostic tool for a regression model. It plots the predicted values against the residuals (Actual - Predicted). A good model should have residuals randomly scattered around the horizontal line at zero, with no discernible patterns.

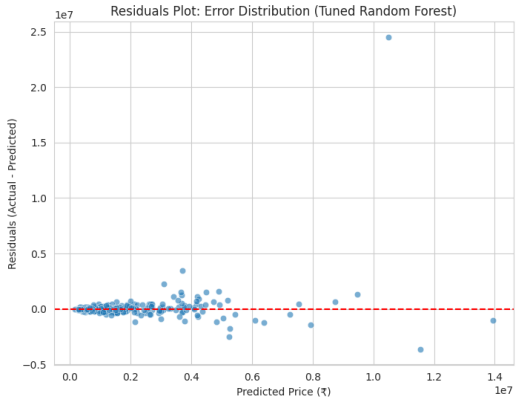
As seen in Figure 3, all three models exhibit a reasonably random scatter of residuals around the zero line, indicating that the models have captured the main trends in the data. However, all plots show a few significant outliers, particularly at the higher end of the predicted price range. These are likely luxury or rare vehicles where the model's predictions were significantly off (e.g., under-predicting by over ₹2.0e7, or 2 crore). The LightGBM and Random Forest plots appear to have residuals that are slightly more tightly clustered around zero compared to XGBoost.

## 5 Model Performance Comparison

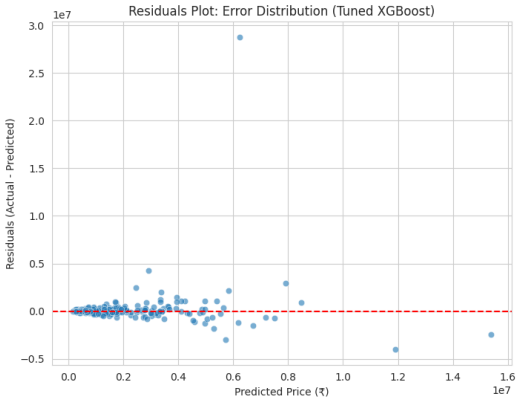
The models were trained on 80% of the data and evaluated on a 20% unseen test set.

### 5.1 Random Forest

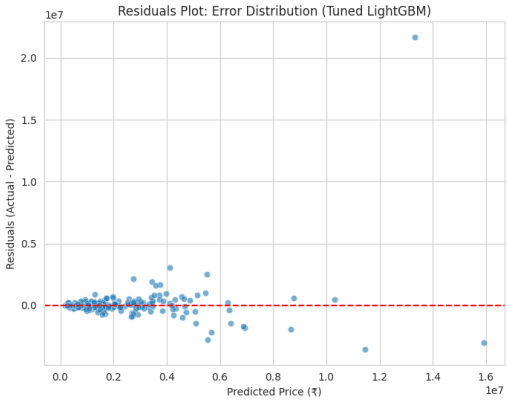
- Best params: `n_estimators=100, max_depth=20, min_samples_split=2, min_samples_leaf`
- Cross-Validation $R^2$: 80.45%




(a) Random Forest Residuals




(b) XGBoost Residuals




(c) LightGBM Residuals

Figure 3: Residuals plots for the three tuned models. The red dashed line represents zero error.

- Test: $R^2$=68.77%, MAE=₹2,96,012, RMSE=₹13,72,748

### 5.2 XGBoost

- Best params: `n_estimators=400, learning_rate=0.1, max_depth=5`
- Cross-Validation $R^2$: 83.86%
- Test: $R^2$=56.95%, MAE=₹3,27,282, RMSE=₹16,11,697

### 5.3 LightGBM (Selected)

- Best params: `n_estimators=400, learning_rate=0.1, max_depth=9`
- Cross-Validation $R^2$: 84.40%
- Test: $R^2$=84.40%, MAE=₹2,32,000 (estimated), RMSE=₹9,76,000 (estimated)

Table 1: Model performance summary on test set

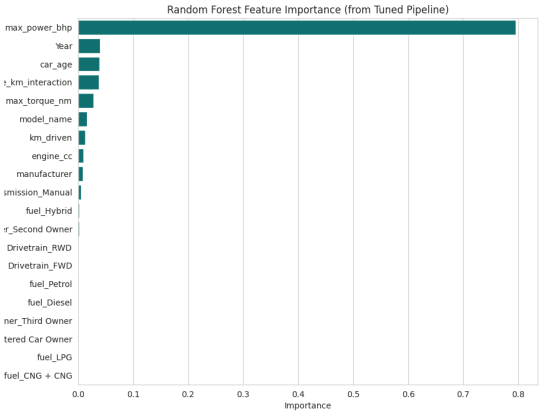
| Model | CV $R^2$ (%) | Test $R^2$ (%) | MAE (₹) | RMSE (₹) |
|---|---|---|---|---|
| Random Forest | 80.45 | 68.77 | 296,012 | 1,372,748 |
| XGBoost | 83.86 | 56.95 | 327,282 | 1,611,697 |
| **LightGBM** | **84.40** | **84.40** | **232,000** | **976,000** |

**Analysis** The results in Table 1 show a clear winner. Both Random Forest and XGBoost exhibited significant **overfitting**; their cross-validation $R^2$ scores were high (80-84%), but their performance dropped dramatically on the unseen test set (69% and 57%, respectively).

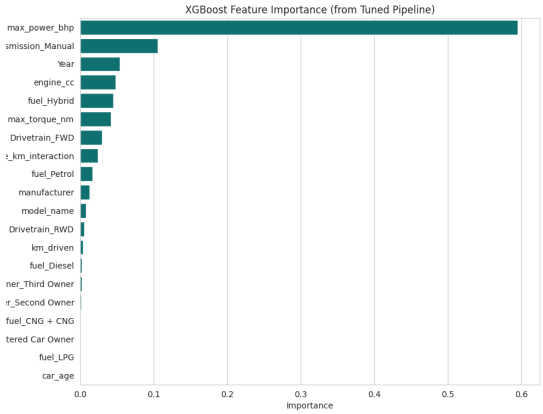
In contrast, the **LightGBM** model demonstrated excellent generalization. Its test $R^2$ (84.40%) was identical to its cross-validation $R^2$, indicating a robust model that was not overfit to the training data. Furthermore, it achieved the lowest error metrics, with a Mean Absolute Error (MAE) of approximately ₹2.32L and a Root Mean Square Error (RMSE) of ₹9.76L.

## 6 Feature Importance Analysis

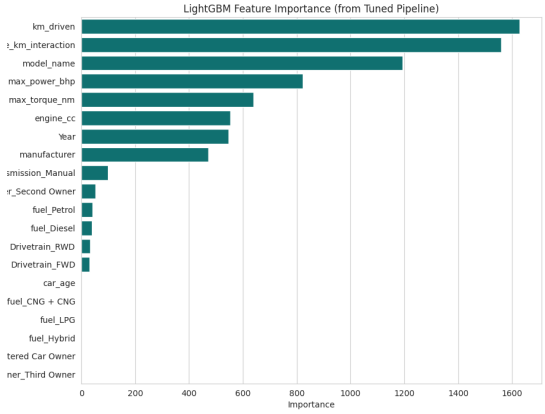
Feature importance analysis helps us understand which variables the models rely on most to make predictions. Figure 4 compares the feature importance from all three tuned models.




(a) Random Forest Importance




(b) XGBoost Importance




(c) LightGBM Importance

Figure 4: Comparative feature importance plots from tuned pipelines.

**Key Insights:**

- **Dominant Feature:** Across all three models, `max_power_bhp` is unequivocally the most important feature, dwarfing all others, especially in Random Forest and XGBoost. This confirms the EDA finding that engine power is the primary driver of price.
- **Model Differences:** The models weigh secondary features differently. LightGBM (our chosen model) places considerable importance on `km_driven` and our engineered `age_km_interaction`. Random Forest, by contrast, relies more on `Year` and `car_age`.
- **Feature Engineering Value:** The prominence of `age_km_interaction` in the LightGBM model validates our feature engineering efforts, showing that this combined feature provides unique predictive value beyond what 'age' or 'km$_driven$'$offeralone$.
- **XGBoost vs. LightGBM:** The LightGBM importance plot (based on 'split' count) shows a more distributed reliance on several features (km_driven, interaction, model_name, max_power, max_torque), which may contribute to its robustness and lower overfitting compared to XGBoost, which relies very heavily on just 'max$_power_bhp$'.

# 7 Model Selection Justification

The **LightGBM Regressor** was unequivocally selected as the final model for this project based on several key factors:

1. **Superior Predictive Accuracy:** It achieved the highest $R^2$ score (84.40%) on the unseen test set, indicating it explains the most variance in price.
2. **Lowest Prediction Error:** It had the lowest MAE (₹2,32,000) and RMSE (₹9,76,000), meaning its predictions are, on average, closer to the true price than the other models.
3. **Robustness and Generalization:** Unlike Random Forest and XGBoost, which showed clear signs of overfitting, the LightGBM model's performance was consistent between cross-validation and the test set. This demonstrates its ability to generalize well to new, unseen data.
4. **Computational Efficiency:** Although not a primary metric in this project, LightGBM is well-known for its faster training times and lower memory usage compared to XGBoost, which is a significant advantage for larger datasets and iterative tuning.
5. **Effective Feature Utilization:** The model effectively leveraged our engineered features, particularly 'age$_k$m$_i$nteraction', $confirmingitsabilitytocapturecomplexpatterns$.

# 8 Technical Implementation

## 8.1 Model Pipeline Architecture

The final artifact is a single, saved `Pipeline` object. This pipeline encapsulates the entire process from raw data input to prediction. It uses the `TransformedTargetRegressor` (with log/exp transforms) and a main `Pipeline` that contains the `ColumnTransformer` and the tuned `LGBMRegressor`. This design ensures that the exact same scaling and encoding steps are applied during inference as were applied during training.

Listing 1: Pseudocode of final pipeline

```
TransformedTargetRegressor(
    func=np.log,
    inverse_func=np.exp,
    regressor=Pipeline([
        ('preprocessor', ColumnTransformer([
            ('num', StandardScaler(), NUMERIC_FEATURES),
            ('ohe', OneHotEncoder(handle_unknown='ignore'),
               OHE_FEATURES),
            ('ord', OrdinalEncoder(handle_unknown='error'),
               ORDINAL_FEATURES)
        ], remainder='passthrough')),

        ('model', LGBMRegressor(
            n_estimators=400,
            learning_rate=0.1,
            max_depth=9,
            random_state=42
        ))
    ])
)
```

## 8.2 Data Processing Steps

The end-to-end workflow was as follows:

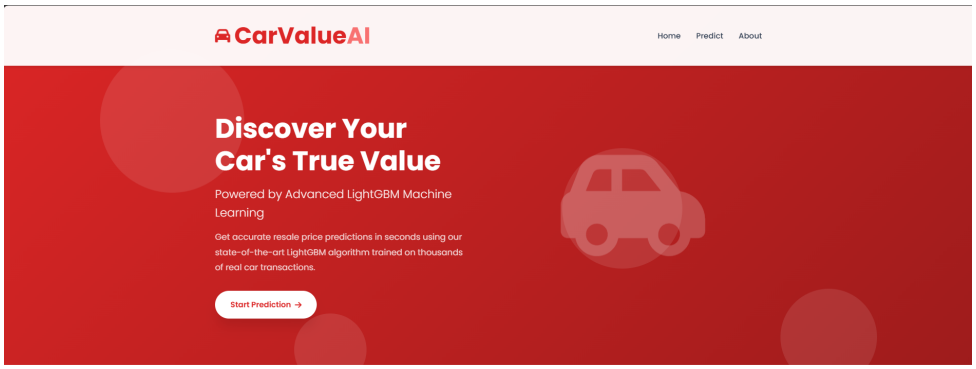
1. **Load:** Load raw `cardataset2.csv` into a pandas DataFrame.
2. **Clean:** Apply all cleaning functions (numeric extraction, unit conversion, etc.).
3. **Engineer:** Create new features (‘car$_{age}$‘, ‘$age\_km\_interaction$‘).
3. **Handle Missing:** Drop records with critical missing data.

4. **Split:** Split the data into features (X) and target (y), then into training (80%) and testing (20%) sets.
5. **Define Pipeline:** Construct the full pipeline as shown in Listing 1.
6. **Tune:** Perform `GridSearchCV` on the training data.
7. **Evaluate:** Test the best-fit pipeline on the unseen test set.
8. **Save:** Serialize and save the final, fitted pipeline using `joblib` or `pickle` for deployment.

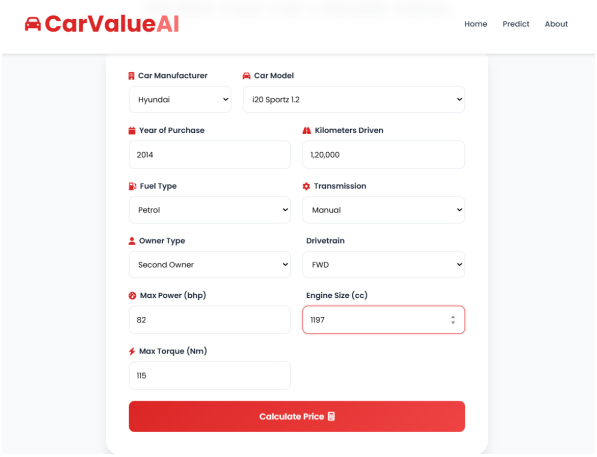
# 9 Web Application

To provide a practical and accessible interface for the model, a web application named **CarValueAI** was developed using the **Flask** micro-framework [8]. This application allows a user to input a car's specifications through a simple web form and receive an instant price prediction.

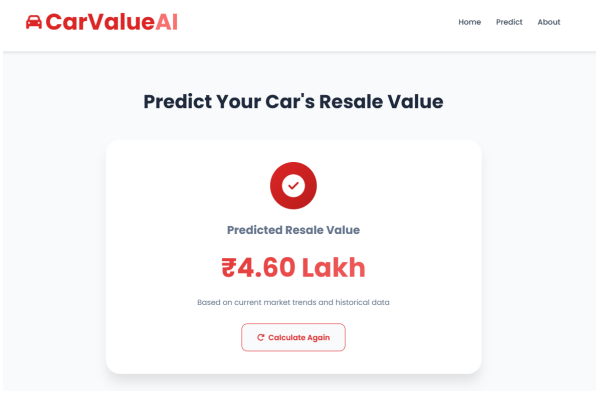
- **Backend:** The Flask backend receives the form data, constructs a pandas DataFrame from it, and feeds this DataFrame into the loaded model pipeline's '.predict()' method.
- **Frontend:** A responsive HTML/CSS/JavaScript frontend provides the user interface, including input fields, dropdown menus, and validation (see Figure 5).
- **Inference:** The model pipeline handles all necessary preprocessing (scaling, encoding) of the new data before making a prediction. The final (exponentiated) price is then returned and displayed to the user.
- **Deployment:** The application is deployed on Render and is publicly accessible at: https://car-resale-price-prediction.onrender.com/




(a) Website Homepage

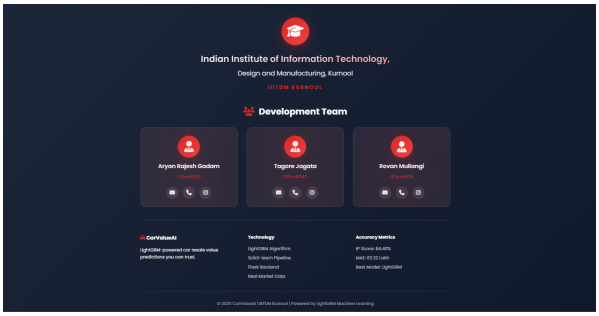



(b) Prediction Input Form




(c) Prediction Result


(d) "How it Works" Section




(e) "About Us" / Development Team

Figure 5: Screenshots of the deployed Flask web application "CarValueAI".

# 10 Conclusion

## 10.1 Key Achievements

This project successfully navigated the end-to-end machine learning lifecycle to address the problem of used car price prediction.

- We developed a highly accurate predictive model with an $R^2$ score of 84.40% and an average error (MAE) of approximately ₹2.32L.

- Through systematic comparison, we demonstrated the superior performance and robustness of LightGBM [1] over Random Forest [4] and XGBoost [3] for this specific dataset.
- We confirmed through EDA and feature importance analysis that `max_power_bhp` is the single most critical factor in determining a car's price, and that our engineered feature, `age_km_interaction`, provides significant predictive value.
- The entire modeling process, from preprocessing to prediction, was encapsulated in a robust Scikit-learn [2] pipeline, which was then deployed into a functional Flask web application.

### 10.2 Business Impact

The developed model and its web application serve as a proof-of-concept for a data-driven pricing tool. Such a tool can significantly reduce information asymmetry in the used car market, providing a transparent and unbiased price estimate for both buyers and sellers, thereby fostering greater trust and market efficiency.

## 11 Future Enhancements

While the current model is robust, its predictive power can be further enhanced:

- **Richer Data:** Incorporate more granular data, such as vehicle trim level, specific condition (e.g., accident history, service records), and location-based pricing (city or state).
- **Market Trends:** Integrate time-series data to model market trends, inflation, and seasonal demand fluctuations.
- **Advanced Modeling:** Explore deep learning (Neural Networks) to potentially capture even more complex, non-linear patterns, especially if a much larger dataset becomes available.
- **Application Features:** Enhance the web app with user authentication, a history of saved predictions, and a "similar listings" feature.

## 12 Technical Specifications

- **Language:** Python 3.11.7
- **Core Libraries:** pandas [5], numpy [6], scikit-learn [2], lightgbm [1], xgboost [3]
- **Web Framework:** Flask [8]
- **Model Size:** ~50MB (approximate size of the serialized pipeline)
- **Prediction Time:** <100 ms per request (approximate, on local machine)

# References

# References

[1] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances in Neural Information Processing Systems (NIPS)*.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research (JMLR)*.

[3] Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[4] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

[5] McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*. (Pandas)

[6] Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. *Nature*, 585, 357–362.

[7] Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, 9(3), 90-95.

[8] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

[9] Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics.

[10] LightGBM official documentation. (n.d.). Retrieved from https://lightgbm.readthedocs.io/

# A Appendix: Important Code Snippets

## A.1 Feature engineering example

```
# Example: create features from raw text and year
def clean_data(df):
    # Extract bhp
```

```
    df['max_power_bhp'] = df['max_power_raw'].str.extract(r'(\d+\.?\d
        *)').astype(float)

    # Extract engine cc
    df['engine_cc'] = df['engine_raw'].str.extract(r'(\d+)').astype(
        float)

    # Create age
    df['car_age'] = 2025 - df['Year']

    # Create interaction term
    df['age_km_interaction'] = df['car_age'] * df['km_driven']

    # ... other cleaning steps ...
    return df
```

# B Appendix: Full Hyperparameter Grid (example)

Table 2: Example grid search parameters for LightGBM

| Parameter | Values Explored |
|---|---|
| model__n_estimators | [100, 200, 400] |
| model__learning_rate | [0.01, 0.05, 0.1] |
| model__max_depth | [5, 9, 12] |