CS5228 Project Report

Amey Shimpi(A0305028N) Manav Chouhan(A0304485A)

Sneha Sarkar(A0304787U) Suryaansh Rathinam(A0307215N)

***Abstract*—The project involved car price prediction using classical machine learning algorithms on a Kaggle dataset. The dataset provided consisted of separate training and test sets and had a diverse set of features which could be utilized for predicting the car price. Initially, the data was processed at a high level by dropping empty feature records or feature columns and filling empty feature values using suitable encoding techniques. This was followed by a more deeper analysis where correlation of each feature was measured with the target and different feature combinations were collated to find the best set of features. Classical Machine learning algorithms like Linear regression, Lasso regression, Random forest regressors and Gradient boosting were used to train the models on top of the processed dataset with different hyper-parameters to find the optimal parameters and the best performing model.**

I. Introduction

The project involved two major sub-tasks; exploratory data analysis and application of machine learning algorithms. The dataset consisted of separate training and test sets. The **initial training data** consisted of **30 columns (features + target)** and **25000 records** whereas the **test data** consisted of **29 columns (only features)** and **10000 records**. The **exploratory data analysis** involved several steps like **dropping of irrelevant features** and usage of **encoding techniques** like **binary encoding** and **target encoding** on categorical features. The **missing feature values** were addressed using **grouped mean** and **overall mean/median imputation** techniques. On the top of existing features, **feature engineering** was performed to calculate relevant features such as ”vehicle age” to further boost our model results. A deeper level of analysis was done by performing a **correlation study** where features with high correlation values were dropped to avoid inflation of model variance and hence prevent overfitting. The **final training set** consisted of **24,999 records** and **25 columns (original features + feature engineered columns)** .Classical machine learning algorithms, such as **linear regression, ridge regression, lasso regression, random forests, and gradient boosting regressors**, were deployed on the final dataset. To optimize model performance, we conducted extensive hyperparameter tuning, using `GridSearchCV` to systematically search through a range of values for each model's key parameters. This approach allowed us to identify the optimal hyperparameters for each algorithm, enhancing predictive accuracy and generalizability.Finally the results of each algorithm were compared to find the overall optimal solution. The plots depicting the feature importance, comparison of actual and predicted values and residual distribution are presented towards the end to gain better insight and enhance explain-ability of the results for the final model.

Best Model:

- **Algorithm:** Gradient Boosting
- **Hyperparameters:** Depth = 5, Number of Estimators = 100
- **Results:** Training RMSE: 10,070.88, Validation RMSE: 25,701.23

II. Exploratory Data Analysis and Data Processing

A. Initial Dataset Overview

We began by examining the structure of the dataset to identify key characteristics, potential issues, and features requiring special handling.

- **Number of Records**: The initial dataset contained 25,000 rows, each representing a unique car listing.
- **Number of Columns**: The dataset had 30 columns, comprising metadata, categorical attributes, and numerical features.
- **Missing Values**: Several columns had substantial missing values:
  - For example, the *fuel_type* column had over 75% missing values, as did columns like *opc_scheme* and *lifespan*.
  - Other columns, such as *depreciation*, *dereg_value*, *mileage*, and *road_tax*, had partial missing values.

B. Dropping Irrelevant Columns

- **listing_id**:
  - **Description**: A unique identifier for each car listing.
  - **Reason for Removal**: This column had no predictive value for car resale prices, as it was merely an ID with no bearing on the car's attributes. Removing it helped reduce data redundancy without affecting the dataset's predictive information.
- **title**:
  - **Description**: A descriptive title for each listing, often containing make and model information.
  - **Reason for Removal**: The *title* column was primarily a free-text field, which could be redundant given the explicit *make* and *model* columns. Dropping *title*

eliminated potential noise and prevented duplication of information already available in structured fields.

- **description**:
  - **Description**: A text description providing additional details about each car listing.
  - **Reason for Removal**: Although this column might contain valuable information in natural language, it was not structured or quantifiable in a way that would easily contribute to a regression model. Removing *description* minimized textual noise, keeping the dataset focused on structured numerical and categorical data.
- **features**:
  - **Description**: Contains information about various specifications or unique features of the car, typically in free text.
  - **Reason for Removal**: As with *description*, this column contained unstructured text data, which was challenging to quantify for regression purposes. After attempting to use this column by utilizing frequency of items, we found it did not improve the results. Dropping *features* further reduced textual noise, helping to maintain a numerical and categorical focus.
- **accessories**:
  - **Description**: Lists any additional car accessories included with each listing, usually as free-text data.
  - **Reason for Removal**: This column was unstructured and not directly linked to resale value in a quantifiable way, similar to *features*. We tried to use it by analyzing item frequency, but it did not improve results. Removing *accessories* contributed to a cleaner dataset with only directly relevant features.
- **original_reg_date**:
  - **Description**: Represents the original registration date of the car.
  - **Reason for Removal**: This column was redundant because car age could be effectively derived from the more complete *reg_date* column. Additionally, it contained 24,744 null values (over 90% of the dataset).
- **fuel_type**:
  - **Description**: Specifies the type of fuel the car uses (e.g., petrol, diesel, electric).
  - **Reason for Removal**: Over 75% of values in this column were missing, making it impractical to retain without significant imputation. Dropping *fuel_type* reduced the dataset's sparsity, allowing us to focus on features with more complete data.
- **opc_scheme**:
  - **Description**: Refers to a specific car ownership scheme that might apply to certain cars in the dataset.
  - **Reason for Removal**: Like *fuel_type*, this column had a high percentage of missing values (24,837 rows) and applied to only a small subset of cars.
- **lifespan**:
  - **Description**: Indicates the expected lifespan of the vehicle under the given ownership scheme.
  - **Reason for Removal**: This column had sparse data and was not directly correlated with resale price, especially since it applied to only specific car types.
- **eco_category**:
  - **Description**: Categorizes the car based on its environmental impact.
  - **Reason for Removal**: This column did not provide distinct categories, as it was largely populated by a single value, making it uninformative for resale prediction. Removing *eco_category* ensured the dataset contained only features with variability, enhancing predictive value.
- **indicative_price**:
  - **Description**: This column was entirely null.
  - **Reason for Removal**: Since all values were missing, the column was removed as it provided no useful information.

C. Handling Duplicates

We checked for duplicate rows in the dataset after removing the *listing_id* column. Duplicates could indicate redundant listings or data entry errors, which could bias the regression model. After dropping *listing_id*, we found and removed one duplicate row at index 18,966.

D. Handling of Categorical Data

To make categorical features compatible with regression modeling, we converted them into numerical representations. Different encoding techniques were applied based on each column's nature to retain meaningful information.

- **Target Encoding**
  - **make**: Specifies the manufacturer or brand of the car. We used target encoding on this column, replacing each make with the average *dereg_value* of cars from that brand, as *dereg_value* had the highest initial correlation with *price* (0.92).
  - **model**: Specifies the specific model within each brand. Similar to *make*, we used target encoding, replacing each model with the average *dereg_value* for that model.
  - **type_of_vehicle**: Specifies the type of vehicle (e.g., sedan, SUV). We applied target encoding to capture resale trends by vehicle type, such as different depreciation rates for SUVs vs. sedans.
- **One-Hot Encoding**
  - **category**: The *category* column contained 16 unique values describing each car, such as *hybrid car* and *imported used vehicle*. We applied one-hot encoding, creating a binary column for each unique category. This approach is effective for limited distinct values, enabling each category to be represented independently in the model.

E. Handling Missing Values

To ensure data consistency and accuracy, we applied different imputation methods depending on the context and nature of each column. Below are the reasons for each imputation approach, organized by the imputation method used.

1) **Grouped Mean Imputation**
   - **Reason**: For columns with values that can vary significantly by model, grouped mean imputation helps retain model-specific trends. By imputing the mean within each model group, we ensure that missing values align with the characteristics typical of each car model, maintaining the unique resale patterns associated with different models.
   - **depreciation** (507 missing values): Reflects the loss in car value over time. Using model-based imputation preserves how different models hold or lose value at various rates.
   - **dereg_value** (220 missing values): Represents the residual value recoverable upon deregistration. Grouped mean imputation ensures that the deregistration value aligns with typical model-specific depreciation patterns.
   - **mileage** (5,304 missing values): Indicates total distance traveled by the car. Imputing by model retains average mileage trends for each model, as different models may have different usage rates.
   - **model_encoded** (derived column): Captures the model's average deregistration value, derived during encoding. Using model-specific averages maintains consistency with how resale trends vary across models.
2) **Overall Mean Imputation**
   - **Reason**: For columns with values that do not vary significantly by model, overall mean imputation provides a straightforward and balanced estimate. This approach standardizes missing values across the dataset, which is appropriate when there are no model-specific patterns to preserve.
   - **road_tax** (2,632 missing values): Reflects the annual tax paid for car ownership, which is fairly consistent across similar vehicle types. Imputing with the overall mean provides a balanced estimate that does not skew based on specific models.
   - **curb_weight** (307 missing values): Represents the car's weight without passengers or cargo. Since weight variations are not strongly model-specific in this context, the overall mean offers a consistent fill value.
3) **Overall Median Imputation**
   - **Reason**: For columns representing discrete or categorical data, median imputation avoids the influence of extreme values and provides a more representative fill value. Median imputation is particularly useful for categorical-like features, as it prevents any skew from outliers.
   - **no_of_owners** (18 missing values): Represents the number of previous owners, a discrete count variable. Using the median avoids the influence of any high or low outliers, ensuring a balanced and realistic fill value for this field.

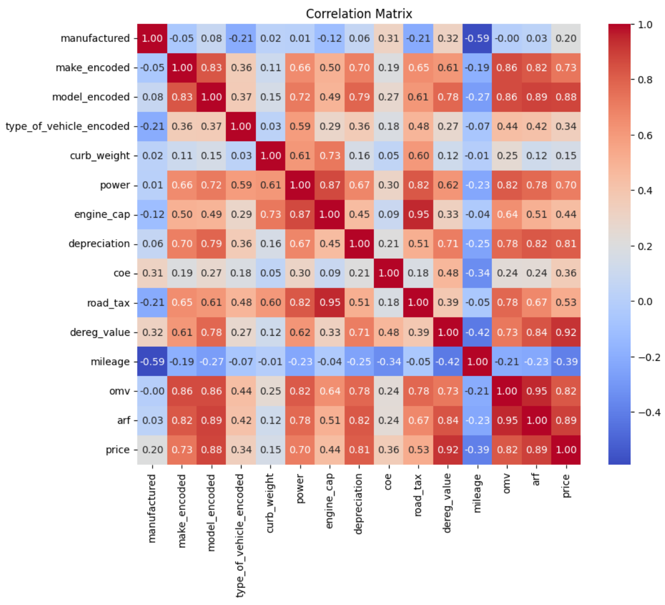
F. Feature Engineering

Feature engineering was performed to enhance the dataset with additional information that could improve the model's ability to predict car resale prices. This process involved deriving new features based on existing columns to capture important factors affecting car value.

1) vehicle_age_months: To capture the impact of a car's age on its resale price, we created a new feature, *vehicle_age_months*, which represents each car's age in months.

- **Method**: We calculated *vehicle_age_months* by taking the difference between the current date and the car's registration date (*reg_date*). This calculation considered both the year and month, providing a precise measure of age.
- **Reason**: Vehicle age is a crucial determinant of depreciation; generally, as a car gets older, its resale value decreases. Representing age in months provides a continuous and granular measure, allowing the model to better capture the subtle effects of time on car value.
- **Outcome**: The *vehicle_age_months* feature replaced *reg_date* in the dataset, providing a straightforward numerical representation of age that is well-suited for regression modeling.

G. Feature Selection and Correlation Analysis



	manufactured	make_encoded	model_encoded	type_of_vehicle_encoded	curb_weight	power	engine_cap	depreciation	coe	road_tax	dereg_value	mileage	omv	arf	price
manufactured	1.00	-0.05	0.08	-0.21	0.02	0.01	-0.12	0.06	0.31	-0.21	0.32	-0.59	-0.00	0.03	0.20
make_encoded	-0.05	1.00	0.83	0.36	0.11	0.66	0.50	0.70	0.19	0.65	0.61	-0.19	0.86	0.82	0.73
model_encoded	0.08	0.83	1.00	0.37	0.15	0.72	0.49	0.79	0.27	0.61	0.78	-0.27	0.86	0.89	0.88
type_of_vehicle_encoded	-0.21	0.36	0.37	1.00	0.03	0.59	0.29	0.36	0.18	0.48	0.27	-0.07	0.44	0.42	0.34
curb_weight	0.02	0.11	0.15	0.03	1.00	0.61	0.73	0.16	0.05	0.60	0.12	-0.01	0.25	0.12	0.15
power	0.01	0.66	0.72	0.59	0.61	1.00	0.87	0.67	0.30	0.82	0.62	-0.23	0.82	0.78	0.70
engine_cap	-0.12	0.50	0.49	0.29	0.73	0.87	1.00	0.45	0.09	0.95	0.33	-0.04	0.64	0.51	0.44
depreciation	0.06	0.70	0.79	0.36	0.16	0.67	0.45	1.00	0.21	0.51	0.71	-0.25	0.78	0.82	0.81
coe	0.31	0.19	0.27	0.18	0.05	0.30	0.09	0.21	1.00	0.18	0.48	-0.34	0.24	0.24	0.36
road_tax	-0.21	0.65	0.61	0.48	0.60	0.82	0.95	0.51	0.18	1.00	0.39	-0.05	0.78	0.67	0.53
dereg_value	0.32	0.61	0.78	0.27	0.12	0.62	0.33	0.71	0.48	0.39	1.00	-0.42	0.73	0.84	0.92
mileage	-0.59	-0.19	-0.27	-0.07	-0.01	-0.23	-0.04	-0.25	-0.34	-0.05	-0.42	1.00	-0.21	-0.23	-0.39
omv	-0.00	0.86	0.86	0.44	0.25	0.82	0.64	0.78	0.24	0.78	0.73	-0.21	1.00	0.95	0.82
arf	0.03	0.82	0.89	0.42	0.12	0.78	0.51	0.82	0.24	0.67	0.84	-0.23	0.95	1.00	0.89
price	0.20	0.73	0.88	0.34	0.15	0.70	0.44	0.81	0.36	0.53	0.92	-0.39	0.82	0.89	1.00

Fig. 1. Correlation Matrix before Feature Selection

To improve model interpretability and reduce redundancy, we performed feature selection based on correlation analysis(Fig 1). The goal was to identify and remove features that provided similar information, which can lead to multicollinearity in regression models. High correlation among features can cause the model to overemphasize certain information, impacting feature importance and increasing variance.

1) Correlation of Features with Target (price): The table below shows each feature's correlation with the target variable, *price*, providing a basis for identifying features with stronger predictive potential:

Feature	Correlation with price
manufactured	0.2029
make_encoded	0.7311
model_encoded	0.8771
type_of_vehicle_encoded	0.3368
curb_weight	0.1521
power	0.7035
engine_cap	0.4431
depreciation	0.8118
coe	0.3565
road_tax	0.5277
dereg_value	0.9196
mileage	-0.3925
omv	0.8221
arf	0.8919

TABLE I
CORRELATION OF EACH FEATURE WITH THE TARGET VARIABLE, *price*.

2) Process and Criteria: **Correlation with Target Variable (price):** We assessed each feature's correlation with the target variable, *price*. Features with a high correlation with *price* were preferred for retention, as they were likely to be more predictive.

Pairwise Correlation Among Features: We calculated pairwise correlations between features to identify highly correlated pairs (correlation coefficient > 0.75). When two features were strongly correlated, we retained the one with a stronger correlation with *price* and removed the other to minimize redundancy.

3) Features Dropped Due to High Correlation or Low Predictive Value: Based on this analysis, we identified the following features for removal:

- **make_encoded**: Dropped due to a high correlation with *model_encoded* (0.8288). While both represent encoded categorical data, *model_encoded* showed a stronger correlation with *price* (0.8771) and was therefore retained to reduce multicollinearity.
- **power**: Dropped as it was highly correlated with *engine_cap* (0.8660) and had a moderate correlation with *price* (0.7035). Retaining *engine_cap* allowed us to reduce redundancy in performance-related features without sacrificing predictive power.
- **omv** (Open Market Value): Dropped due to high correlation with *arf* (0.9463) and a lower correlation with *price* (0.8221). Retaining *arf* simplified the model while keeping features with stronger predictive power.
- **engine_cap**: Dropped because of high correlation with *road_tax* (0.9468) and a relatively lower correlation with *price* (0.4431). Retaining *road_tax* captured the financial implications of engine capacity without redundancy.
- **arf** (Additional Registration Fee): Dropped due to a high correlation with *dereg_value* (0.8374) and a lower correlation with *price* (0.8919). Keeping *dereg_value*, which has a stronger relationship with *price* (0.9196), reduced redundancy.
- **curb_weight**: Dropped as it showed minimal correlation with *price* (0.1521), indicating low predictive value.
- **coe** (Certificate of Entitlement): Dropped due to its low correlation with *price* (0.3565), contributing little to the model's predictive power.
- **type_of_vehicle_encoded**: Dropped because of its low correlation with *price* (0.3368), making it a weak predictor.

H. Final Dataset Summary

After completing the EDA and data preprocessing steps, we arrived at a high-quality dataset with the following characteristics:

- **Number of Columns**: The final dataset contains 25 columns, focused on features that directly relate to car specifications, conditions, and pricing factors.
- **Number of Rows**: After cleaning, we retained 24,999 records, removing any duplicate entries.
- **Missing Values**: No missing values remain in the final dataset, ensuring consistency across all records.

III. INCONCLUSIVE EXPLORATORY DATA ANALYSIS

A. Manufactured and Registration Date

The `manufactured` and `reg_date` features are nearly identical, differing by only 2–3 years at most. Additionally, manufactured has 7 null values, whereas `reg_date` has none. We explored the possibility of filling the missing manufactured values with the corresponding year from `reg_date` where applicable.

The manufactured and `reg_year` fields match in 15,027 records and differ in 9,972 records, though the difference is typically only one or two years. Given this close alignment, we imputed missing values in manufactured using `reg_year`, as it represented the majority pattern. However this approach did not make much of a difference towards our model accuracy.

B. Make and Model

There were two columns: `make` and `model`. The `model` corresponds to the car model for the given `make`. While some records have a null value in the `make` column, there are no records with a null value in the 'model' column. Since each `make` is associated with unique models, it was considered whether the `make` could be reverse-engineered from the 'model' in cases where the `make` is missing.

To reverse engineer the `make` from the `model` in the dataset, the following steps were performed:

1) **Create a Mapping for `make-model` Pairs**: We created a mapping of `make` to the associated `model` values using the training data. This mapping helps identify which models are associated with which makes. The training data was filtered to exclude rows with missing

make values. The data was then grouped by make, and a list of models for each make was generated.

2) **Filling Missing make Values**: We used the mapping from the previous step to fill in missing make values in the test dataset based on the model column. For each row with a missing make, the corresponding model was checked against the list of models for each make. If the model matched any model in a make's list, the corresponding make was assigned to the missing value.
3) **Apply the Filling Logic**: The filling function was applied row by row to the dataset to fill in the missing make values. This was done for the test dataset using the mappings derived from the training data.

These steps ensured that missing make values were inferred from the model column using a mapping derived from the training data.

This kind of logic could not be applied to the test dataset, as it also contained missing values and lacked the corresponding necessary information. Additionally, it was considered that reverse-engineering the columns might lead to high correlation between features.

C. Curb_Weight

For the curb_weight column, the following steps were initially considered:

1) **Averaging curb_weight by make**: Initially, we attempted to handle the missing values in the curb_weight column by averaging the curb_weight values within each make category. This would allow us to fill the missing values with the average curb_weight for that particular make, assuming that cars of the same make had similar curb weights.
2) **Visualization and Realization**: Upon visualizing the data, it became clear that there is a significant range of curb_weight values within each make. This variability indicated that using the average curb weight for each make might not be appropriate, as different models within the same make could have vastly different curb weights.
3) **Conclusion**: Due to the variability in curb_weight within each make, filling the missing curb_weight values with the mean value of the make would not be an accurate or reasonable approach. Therefore, we decided against using the average curb_weight.

This process highlighted the complexity of using aggregated values (like the mean) for certain columns when there is high variability in the data, especially for columns like curb_weight that can vary significantly even within a single make.

D. Fuel Type

For the fuel_type column, the following approach was attempted:

1) **Initial Assumption Based on reg_year**: Initially, we hypothesized that the fuel_type might be related to the reg_year, with older cars likely having diesel or petrol as their fuel type, while newer cars might be more likely to be electric or hybrid. This assumption was based on trends where newer vehicles are shifting toward more eco-friendly fuel types.
2) **Visualization**: We analyzed the data by looking at the distribution of fuel_type across different reg_year values. This allowed us to observe if there were clear patterns, such as an increasing prevalence of electric or hybrid vehicles in recent years, while diesel and petrol dominated the older vehicles.
3) **Outcome and Conclusion**: However, there was no trend between years and fuel_type.

This approach highlighted the utility of reg_year as a proxy for determining fuel_type, but also underscored the importance of exploring each column's specific distribution before making assumptions for imputation.

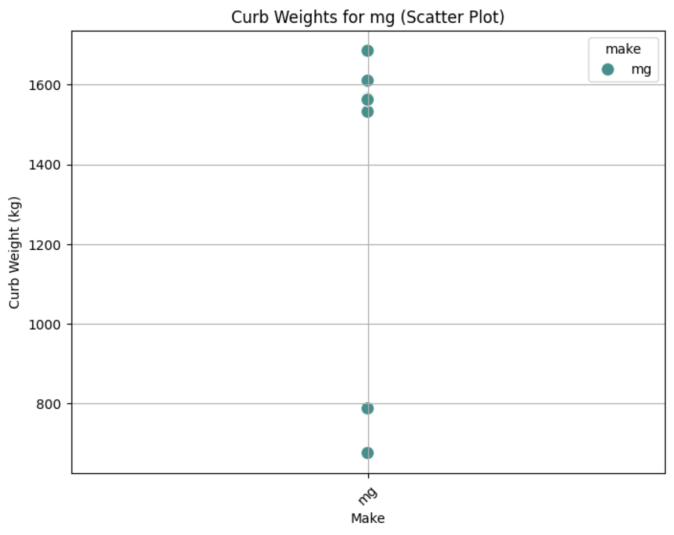


Fig. 2. Curb Weights Grouped by a Specific Make 'mg'

E. Lifespan

This column has null for 22670 rows. We initially hypothesized that for cars that have exceeded their lifespan the price should be 0, as the car would be deregistered and so it would not be viable for reselling.

However, even cars that have exceed their lifespans have a reselling price so this logic was not viable.

F. Features, Description and Accessories

These were pure text string columns. Here is the approach for the accessories column, a similar approach was followed for the description and features columns as well:

1. We converted accessories to a string type, which ensures that any non-string values (e.g., NaN or lists) are treated as strings for further processing, such as splitting and checking substrings.

2. The values are split at commas. The `explode()` function is then applied to transform each into an individual row, effectively flattening the column.

3. Any leading or trailing spaces from the accessory names are removed using the `strip()` method to ensure consistency and prevent issues in counting or matching accessory names.

4. The `Counter` class from the `collections` module is used to count how often each `accessories` appears across all rows. The result is a dictionary, where each key represents an accessory, and the value represents the number of times it appears in the dataset.

5. The dictionary of `accessories` counts is converted into a pandas Series, and the `nlargest(6)` method is used to return the top six most frequent accessories in the dataset. These accessories are chosen as the focus for creating new binary features.

6. A list of the top accessories is manually defined, excluding 'nan' as it represents missing data rather than a valid accessory. These are the accessories that will be used to create binary columns in the dataset.

7. For each of the top accessories, a new binary column is created in the dataset. The column indicates whether a particular accessory is present in the `accessories` column. A value of 1 indicates that the accessory is present, while 0 indicates its absence.

Although commonly repeated features, descriptions, and accessories were identified and their correlation with price was plotted, they did not exhibit particularly high correlations. We also tried splitting by comma and counting how many items are there in each and encode them with that value i.e. higher number of features / accessories would result in higher prices but that was also not the case.

G. Outlier Removal

The outlier detection process was performed on the 'price' column of the dataset using the Interquartile Range (IQR) method. The first (Q1) and third (Q3) quartiles were calculated, and the IQR was computed as the difference between Q3 and Q1. Outliers were defined as values falling outside the range of 1.5 times the IQR below Q1 or above Q3 as shown in the figure below. However, after applying this outlier removal technique, the dataset shape does not change much, with the resulting number of records at 23,217.

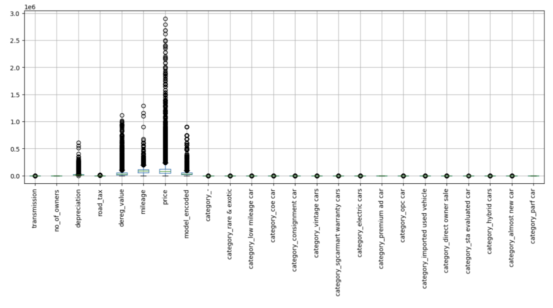


Fig. 3. Number of Outliers per column

IV. Results and Discussion

In this section, we present and analyze the outcomes of our predictive modeling efforts, focusing on the performance of various regression models applied to our pre-processed dataset. Prior to model training, the input data was standardized using `StandardScaler` to ensure uniform feature scaling. Additionally, we employed 10-fold cross-validation to enhance the robustness of our model evaluation, providing a comprehensive assessment of model accuracy and stability across different subsets of the data.

A. Evaluated Models and Performance

We explored five regression models to identify the most suitable predictor for our data. Each model's configuration and performance are detailed below:

1. Linear Regression: This basic regression model fits a linear relationship between independent variables and a continuous target variable.

- **Hyperparameters:** Default settings.
- **Performance:**
  - Training RMSE: 45,773.67
  - Validation RMSE: 45,953.31

2. Ridge Regression: An adaptation of linear regression that incorporates L2 regularization to control overfitting by penalizing large coefficients.

- **Hyperparameters:** `alpha` = 1.0 (regularization strength).
- **Performance:**
  - Training RMSE: 45,773.69
  - Validation RMSE: 45,953.08

3. Lasso Regression: Similar to Ridge but utilizes L1 regularization, which can eliminate non-significant features, effectively performing feature selection.

- **Hyperparameters:** `alpha` = 0.01 (regularization strength).
- **Performance:**
  - Training RMSE: 45,773.67
  - Validation RMSE: 45,953.31

4. Random Forest Regressor: This ensemble model aggregates the predictions of multiple decision trees to improve predictive accuracy and reduce overfitting.

- **Hyperparameters:** `n_estimators` (number of trees).
- **Performance:**
  - `n = 50`: Training RMSE: 10,476.58, Validation RMSE: 26,684.85
  - `n = 100`: Training RMSE: 10,215.15, Validation RMSE: 26,200.05
  - `n = 200`: Training RMSE: 9,981.83, Validation RMSE: 26,075.01

5. Gradient Boosting Regressor: This model builds a series of trees sequentially, with each new tree attempting to correct the errors of the previous ones.

- **Hyperparameters:** `max_depth` (tested values: 3, 5, 8, 12), `n_estimators` = 100.

- **Performance:**
  - Depth = 3: Training RMSE: 20,817.35, Validation RMSE: 28,813.67
  - Depth = 5: Training RMSE: 10,070.88, Validation RMSE: 25,701.23
  - Depth = 8: Training RMSE: 3,134.33, Validation RMSE: 27,579.13
  - Depth = 12: Training RMSE: 839.16, Validation RMSE: 29,569.77

B. *Result Analysis and Model Insights*

Linear Models: Linear Regression, Ridge, and Lasso performed similarly, with high RMSE values indicating underfitting. The similarity in training and validation RMSE suggests that these models did not capture the data's complexity.

- **Reasoning:** Linear models presuppose a linear relationship between features and the target variable. When the actual relationship is nonlinear, these models often fail to perform adequately.

Random Forest Regressor: The Random Forest models showed substantial improvements over linear models. Increasing the number of estimators reduced both training and validation RMSE, with diminishing returns beyond `n = 100`.

- **Reasoning:** Random Forests average across multiple trees, which lowers variance and enhances generalization compared to single decision trees.

Gradient Boosting Regressor: Gradient Boosting exhibited the best performance among all models, especially with a maximum depth of 5, yielding the lowest validation RMSE.

- **Optimal Configuration:** Depth = 5 yielded a balance between capturing data complexity and avoiding overfitting.
- **Reasoning:** Gradient Boosting sequentially corrects previous errors, so a moderate tree depth can help manage bias and variance.

C. *Final Model Selection*

After evaluating RMSE across all models, the Gradient Boosting Regressor with a max depth of 5 and 100 estimators was selected as the final model.

- **Optimal Validation Error:** It achieved the lowest validation RMSE (25,701.23), suggesting strong generalization to unseen data.
- **Bias-Variance Balance:** A max depth of 5 effectively balances complexity and overfitting.
- **Stability:** Gradient Boosting is robust to outliers and can capture complex relationships in the data.

D. *Visual Representations of the Final Model*

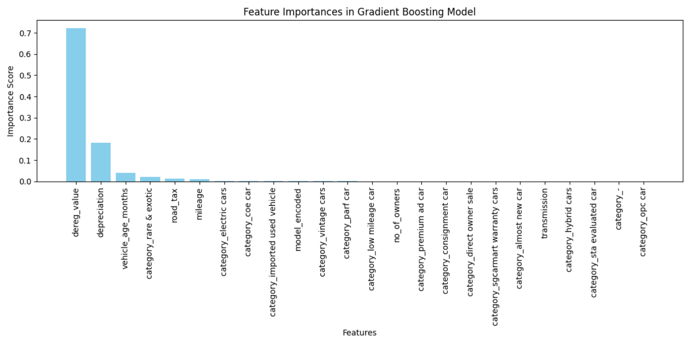


Fig. 4. Feature Importances in Gradient Boosting Model

This bar plot highlights the importance of features, with variables like "dereg_value" and "depreciation" having the highest influence on the target variable, while others contribute less to the prediction.

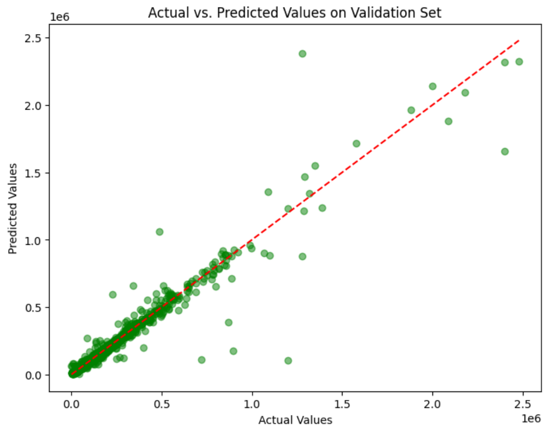


Fig. 5. Actual vs. Predicted Values on Validation Set

The scatter plot compares actual versus predicted values, with points close to the diagonal line reflecting accurate predictions. The spread offers insights into prediction accuracy, with a tighter distribution indicating a better fit.

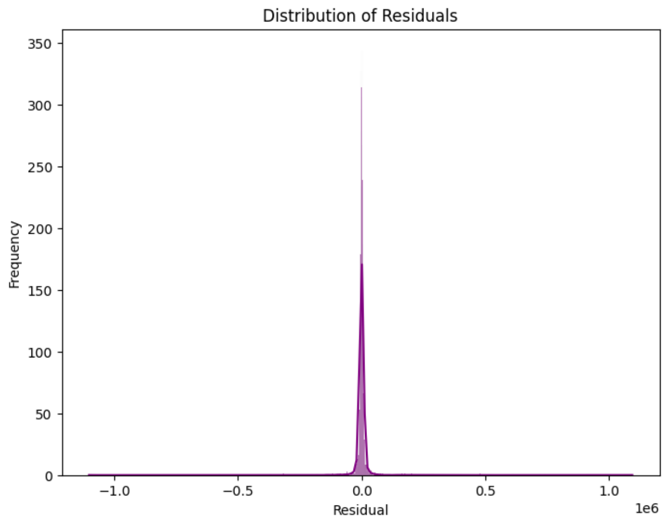


Fig. 6. Distribution of Residuals

This histogram of residuals illustrates that errors are roughly normally distributed around zero, suggesting an unbiased model with consistent performance across the dataset.

In conclusion, the Gradient Boosting Regressor with a depth of 5 provides the optimal balance between model complexity and generalization ability, achieving the lowest RMSE and effectively capturing the intricacies of our data. These results lay a solid foundation for future predictive analyses and potential enhancements in model tuning and feature engineering.

V. Conclusion

This project successfully developed a predictive model for car prices using classical machine learning techniques on a Kaggle dataset. Key steps included data preprocessing, feature engineering, and model selection, with Gradient Boosting Regressor emerging as the most effective model.

- **Data Processing and Feature Engineering:** We carefully handled missing values, encoded categorical features, and engineered relevant features such as ”vehicle age.” Unnecessary and highly correlated features were removed to improve model efficiency and accuracy.
- **Model Performance:** Gradient Boosting, with a max depth of 5 and 100 estimators, achieved the best validation RMSE of 25,701.23, outperforming linear models and random forests in capturing complex relationships in the data.
- **Insights:** Feature importance analysis identified ”dereg_value” and ”depreciation” as key predictors, and residual analysis indicated an unbiased model fit.

In summary, Gradient Boosting provided a robust, interpretable solution for car price prediction, demonstrating the importance of effective feature engineering and model tuning in predictive accuracy.

VI. Appendix

A. Team Contributions

The completion of this predictive modeling project was made possible by the combined efforts of our team, with members collaborating on different aspects to ensure a comprehensive approach to data analysis, modeling, and reporting:

- **Suryaansh Rathinam(A0307215N):** Worked on data collection and initial data cleaning, addressing missing values and encoding categorical features. Conducted exploratory data analysis to identify significant patterns and trends, and worked closely to select relevant features for modeling.
- **Manav Chouhan(A0304485A):** Collaborated on data pre-processing, including scaling features with `StandardScaler` and engineering new potential features to improve model performance. Also supported in preparing the data pipeline for efficient model training and evaluation.
- **Sneha Sarkar(A0304787U):** Focused on model selection, training, and tuning for Linear, Ridge, Lasso, Random Forest, and Gradient Boosting models. Also worked on hyperparameter optimization, employing 10-fold cross-validation to ensure reliable model performance metrics across all models.
- **Amey Shimpi(A0305028N):** Analyzed model performance, comparing RMSE values and ensuring that results were robust. Created visualizations for feature importance, actual vs. predicted values, and residual distributions. Additionally, compiled and authored the report, working with other team members to document each phase of the project and manage code on GitHub.

B. Kaggle Team and GitHub Repository

- **Kaggle Team Name:** MASS
- **GitHub Repository:** https://github.com/suryaansh2002/CS5228-Final-Project