

Due: Sunday, Nov 16, 11:59PM

Welcome to HW3! This assignment features a Kaggle competition on car price prediction, where you'll engineer effective features and apply the machine learning models. The homework consists of a prediction competition and a technical report.

Essential Resources:

- Tree Models in ISLP: <https://islp.readthedocs.io/en/latest/labs/Ch08-baggboost-lab.html>
- Kaggle Competition Guide: <https://www.kaggle.com/learn/guide/kaggle-competitions>
- Feature Engineering: <https://www.kaggle.com/learn/feature-engineering>

Recommended Workflow:

- Start with EDA and basic data cleaning
- Implement a baseline model (e.g., decision tree, GAM) and get used to submitting your results
- Gradually explore advanced models (Random Forest, XGBoost) and ensemble them
- Use cross-validation to prevent overfitting
- Document your progress and insights

Deliverables:

- Submit a single PDF via Gradescope ("HW3 Write-Up")
- Include your competition results and technical report
- Attach your code in the appendix
- If you use any external dataset for training, please write the link to the dataset.
- Use LaTeX template provided (or neat handwriting if necessary)

Important Guidelines:

1. Sign the honor code statement on the next page
2. Document any collaboration or help received
3. Write all responses in English
4. Properly mark question sections in Gradescope

For staff use only

Q1 (Part 1)	Q2 (Part 2)	Bonus	Total
/ 40	/ 50	/ 10	/ 90

Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature: Jumyung Park

We welcome group discussions, but the work you submit should be entirely your own. If you use any information or pictures not from our lectures or readings, make sure to say where they came from. Please note that breaking academic rules can lead to severe penalties.

- (a) Did you receive any help whatsoever from anyone in solving this assignment? If your answer is 'yes', give full details (e.g., "Alex shared insights on optimizing hyperparameters for XGBoost during a group discussion.")

No

- (b) Did you give any help whatsoever to anyone in solving this assignment? If your answer is 'yes', give full details (e.g., "I advised Chris to check out the Kaggle feature engineering guide for handling missing values.")

No

- (c) Did you find or come across code that implements any part of this assignment? If your answer is 'yes', give full details (book & page, URL & location within the page, etc.).

I followed the feature engineering tutorial on <https://www.kaggle.com/learn/feature-engineering>, which was listed in the essential resources.

Q1. Mini Competition: Car Price Prediction [100 pts]

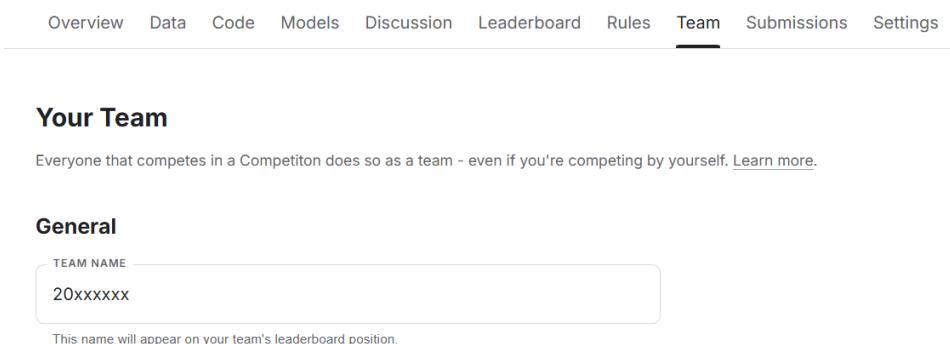
In this mini-competition, you'll work with real-world data to build and improve your predictive models. Your journey will be evaluated on two aspects: your model's performance on the Kaggle leaderboard (40 points) and a technical report documenting your approach (50 points). Bonus points will be awarded as follows: 5 points for the best reports and 5 points for the best competitors.

Part 1: Competition Performance [40 pts]

Your task is to predict car prices. Strive to surpass the target score and submit your results. You're free to use any libraries available - such as scikit-learn, pandas, numpy, statsmodel, ISLP, xgboost, polars and others. Your grade for this part will be determined entirely by your final performance on the private leaderboard. Follow these simple rules: Use your student ID, play fair, give it your all, and, most importantly, enjoy the mini-competition: <https://www.kaggle.com/t/7fb5c52dc7bee6f4b9eff6a4e40bcd83>

Score guideline

- The leaderboard uses Mean Absolute Percentage Error (MAPE) as the evaluation metric, where smaller scores indicate better models.
- While you'll see and capture your public leaderboard score during development, final grading will be based on the private leaderboard score to ensure your model generalizes well to unseen data.
- You can submit up to five times per day. We encourage you to divide the dataset into train/validation sets and use your internal validation set to select the best model for submission.
- Final points will be determined by your private leaderboard score s with the target score t .
- Who achieved better than the target score ($s \leq t$) will receive the full points (40 pts).
- Those who didn't beat the target score ($t < s$) will receive $40 * \min((t/s), 1)$ points.
- You're free to use the Ed & Kaggle discussion tab for sharing your thoughts or your code snippets.
- Referencing or reusing code from other students, making multiple accounts will be treated as an honor code violation. (Honor code at <https://sundong.kim/courses/mldl25f>)



The screenshot shows the 'Team' tab selected in the navigation bar. Below it, the 'Your Team' section is displayed. It includes a note about competing as a team, a 'General' section with a 'TEAM NAME' input field containing '20xxxxxx', and a note that this name appears on the team's leaderboard position.

Figure 1: Use your student ID as a team Name

Part 2: Technical Report [50 pts]

Write a self-contained report (at most 5 pages) documenting your approach and findings. For reference on how to structure your report, you can check example write-ups at <https://wsdm-cup-2018.kkbox.events/>. The report should comprehensively document all stages of your analytical process and methodology.

Your report should include:

- **Problem Description and Data**

- Clear statement of the task and evaluation metric
- Dataset characteristics and initial insights
- Train/validation split strategy

- **Data Preprocessing & Feature Engineering**

- Data cleaning and handling of missing values/outliers
- Feature creation rationale and importance analysis
- Feature selection process

- **Modeling & Results**

- Model architecture and validation methodology
- Hyperparameter tuning strategy
- Performance analysis and key findings

- **Technical Implementation**

- Key libraries/frameworks used
- Challenges faced and solutions

Format requirements:

- Please limit the main document to 5 pages in the current single-column LaTeX format, with unlimited references allowed beyond the page limit. (Use the template at the end of this document; longer submissions will be penalized significantly)
- Include a screenshot of your public leaderboard score as evidence
- Include representative code snippets for important steps
(Data preprocessing, feature engineering, model training)
- Include relevant figures/tables to support your discussion
- Proper citations if you reference any external resources
- Clear structure with appropriate section headers
- Use clear and concise technical writing

Assessment Guidelines

- **Data Type Conversion**

- Correctly convert string features to numeric (e.g., "20.0 kmpl" → 20.0)
- Handle multiple units within the same feature
- Document your conversion strategy and ensure consistency

- **Missing Values and Outliers**

- Thoroughly check for missing values in all columns
- Document your strategy for handling missing data
- Document and justify your handling strategy for missing values
- Identify and appropriately handle outliers with justification

- **Data Splitting Strategy**

- Explain your choice of split ratio and methodology

- **Feature Engineering**

- Create meaningful derived features based on domain knowledge
- Analyze feature importance and select relevant features
- Document the rationale behind each engineered feature

- **Model Training and Evaluation Metrics**

- Justify your choice of training objective and explain the relationship between training metric and evaluation metric
- Report how you conduct a comprehensive performance assessment on your validation set.

- **Hyperparameter Tuning**

- Manually search for optimal hyperparameters (grid search or random search)
- Show iterative improvements through parameter tuning

- **Prohibited Tools/Methods**

- Automated hyperparameter tuning libraries (e.g., Optuna, Hyperopt, Ray Tune)
- AutoML frameworks (e.g., Auto-sklearn, TPOT, H2O AutoML, PyCaret's automated comparison)

Required Evidence of Manual Work:

To demonstrate genuine understanding and ensure academic integrity, your report must include:

1. Experimental Process and Comparisons

- Document your iterative development showing progressive improvements
- Compare multiple approaches systematically:
 - Different models (e.g., Random Forest vs XGBoost)
 - Different preprocessing strategies (outlier handling, target transformation)
 - Different strategies for high missing rate features
- Present results in comparative tables or visualizations
- Hyperparameter Search Documentation
 - Provide a table showing at least 3 different hyperparameter combinations you tested
 - Include validation performance for each configuration

Example table format:

Config	n_estimators	max_depth	learning_rate	MAPE
1	100	5	0.1	0.38
2	200	5	0.1	0.37
3	200	7	0.1	0.36
4	200	7	0.05	0.25
5	300	7	0.05	0.20

Table 1: Example of documented hyperparameter search process

2. Feature Engineering Justification

- For each significant engineered feature:
 - *Motivation*: Why you created it
 - *Implementation*: How you computed it (with code)
 - *Impact*: Performance change after adding it
- Example: “Created booking_days_advance = departure_date - booking_date because early bookings are typically cheaper. Correlation: -0.64, improved MAPE from 0.32 to 0.25.”

3. Code Transparency

- Include code snippets for key steps with manual implementation
- Comment your code to explain design choices
- Avoid showing only high-level library calls without context

If you are uncertain whether a specific tool or method is allowed, please consult with staffs before using it.

Part 3: Bonus Points [10 pts]

Outstanding submissions that demonstrate exceptional quality and performance can earn up to 5 bonus points.

Bonus Point Categories:

- **Exceptional Report Quality [up to 5 pts]**
 - Top 10 reports demonstrating exceptional analysis and documentation
 - Selected based on:
 - * Depth and breadth of experimental comparisons
 - * Quality and clarity of explanations and visualizations
 - * Insightful analysis of results, including thorough error analysis
 - * Professional presentation with well-structured narrative
 - * Evidence of critical thinking and creative problem-solving
- **Outstanding Leaderboard Performance [up to 5 pts]**
 - Top 10 submissions on the final private leaderboard (based on MAPE)
 - Demonstrates effective modeling and feature engineering in practice

Important Notes:

- Students who qualify for both categories (exceptional report and top 10 leaderboard) will receive 10 bonus points in total
- Bonus points are awarded at the instructors' and TAs' discretion
- Focus on demonstrating thorough understanding and thoughtful experimentation rather than solely pursuing the highest score
- Quality of explanation and justification matters as much as final performance

What Makes a Report Exceptional:

Reports that stand out typically include:

- Multiple well-designed experiments with clear comparisons
- Thoughtful handling of challenging aspects (e.g., high missing rate features, outliers, etc..)
- Clear documentation of the iterative development process
- Professional visualizations that effectively communicate insights
- Honest discussion of what worked and what didn't, with analysis of why
- Evidence of going beyond minimum requirements with creative approaches
- Clear, well-organized writing that tells a coherent story

Used-Car Price Prediction

Kaggle GIST-MLDL-25F-HW3 Competition

Jumyung Park #20235073

School of EECS, Gwangju Institute of Science and Technology

1 Task and Data

1.1 Task

The task of this competition (GIST-MLDL-25F-HW3 [1]) is to predict the price of a used car. Performance is evaluated with Mean Absolute Percentage Error (MAPE) between the predicted price and actual price. Training dataset is public while test dataset's target variable is private. The prediction will be evaluated on MAPE measure on the hidden test dataset target variable.

1.2 Dataset

The dataset consists of features listed in Table 2. Total 4470 training samples are provided and test dataset consists of 1491 rows, without the target variable Price. Target variable is numerical, thus the task is a regression problem of predicting the numerical value of Price with all the other features. Features contain both categorical and numerical values. Some features have missing values and mixed units. These inconsistencies were addressed in data preprocessing.

Table 2: Data description

Feature	Data Type	Explanation
ID	str	Unique identifier for each car listing
Name	str	The brand and model Name of the car
Location	str	The city where the car is being sold
Year	int	Manufacturing year of the vehicle
Kilometers_Driven	int	Total distance the car has been driven (in kilometers)
Fuel_Type	str	Type of fuel the car uses (Petrol, Diesel, CNG, LPT, Electric)
Transmission	str	Type of transmission system (Manual or Automatic)
Owner_Type	str	Number of previous owners (First, Second, Third, Forth & and Above)
Mileage	str	Fuel efficiency of the car (kmpl or km/kg)
Engine	str	Engine displacement in cubic centimeters (CC)
Power	str	Maximum power output of the engine (bhp - brake horse power)
Colour	str	Exterior color of the vehicle
Seats	int	Number of seating capacity
No. of Doors	int	Number of doors in the vehicle
New_Price	str	Original price of the car when it was new
Price	float	Target variable: Current selling price of the used car (in lakhs)

2 Data Preprocessing and Feature Engineering

2.1 Data Preprocessing

The dataset was thoroughly inspected feature by feature to spot any inconsistencies in representations and remedy them so that any further processings down the line does not get unexpected errors. Multiple issues in representation was found in the process.

- Missing values denoted with the string \\N were replaced with None.
- New_Price was in mixed units (Lakh and Cr). It was parsed and converted to Lakh unit.
- Mileage was in mixed units (kmpl and km/kg). It was parsed and converted to kmpl unit.
- Price and New_Price had non-negligible amount of missing values (each 2.37% and 86.31% of training dataset). This had to be carefully addressed in feature engineering.
- For the processing on dataframes on the following processes, each features were converted to ‘category’ or their respective numerical datatypes after standardizing units and parsing.

Unit standardization in New_Price was straight forward since Lakh and Cr were just different scales of the same measure ($1\text{ Cr} = 100\text{ Lakh}$). However, unit standardization in Mileage were between two different measures. Liters is a measure of volume and Kg is a measure of mass. Therefore, Fuel_Type was used in the conversion to determine the density to use in the conversion. Fuel density for each fuel type was used in this preprocessing as an external knowledge. Since Fuel_Type had no missing values in both training and test dataset, the conversion factor derivation from it was available for all given Mileage values. Details in the process is documented in the notebook code attached as Appendix A.2

2.2 Feature Engineering

2.2.1 Systematic Feature Engineering Process

Given the preprocessed data, feature engineering was performed systematically to remove noise and extract information as much as possible. The purpose of this systematic process was to not miss any processings that can be done and make data-driven decisions on whether to adopt or discard the attempted engineering. It is essentially a manual greedy forward selection with cross validation. This systematic approach turned out to be effective in building the optimal pipeline while trying numerous approaches in scale, which led to 0.12191 MAPE score on public leaderboard without any tuning on the model. In fact, around 30 transformations were attempted and 13 of them were adopted to form the final feature engineering pipeline. Some of the non-trivial feature engineerings are highlighted in the following subsections. The entire feature-engineering process is documented in the notebook in Appendix A.3.

2.2.2 Feature-by-Feature Engineering

Name Splitting and Target Encoding Name is string feature consisting of Brand and Model. Since Name has large cardinality, it was split into Brand and Model. For robustness, an enum of recognized brands was constructed and used for extracting Brand and Model from Name. Since Brand and Model have a hierarchical relationship, various methods including dropping Model, dropping Name, target encoding Model, Brand, or Name, or all of them were attempted. Among those, including and target encoding Model, Brand, and Name yielded the biggest gain compared to the established baseline with only preprocessing. Exploiting the hierarchical relationship, creating Model_Premium feature to represent tier within the Brand was attempted, but discarded due to no visible gain of performance. Engineering on Name turned out to be one of the most critical process, resulting in -0.0101 reduction in cv-score compared to the baseline.

Power Imputation Power feature had non-negligible amount of missing values (2.37% of training data, and 1.95% of test data). Utilizing the domain knowledge that the engine displacement is directly related to its power in combustion, I inspected its correlation with Engine and it showed high correlation of 0.86. Exploiting this correlation, PowerImputer was constructed by fitting a linear regression on Power to Engine and using it to interpolate the missing Power value. Fortunately, there were dense enough samples around the missing Power values and thus could be faithfully inferred. This imputation led to 0.0055 reduction in MAPE score, compared to the baseline without any feature engineering.

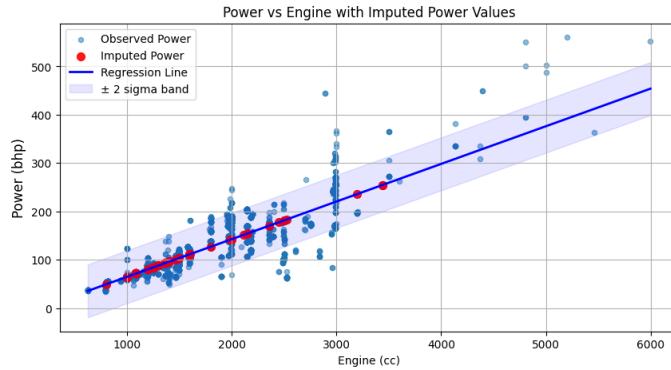
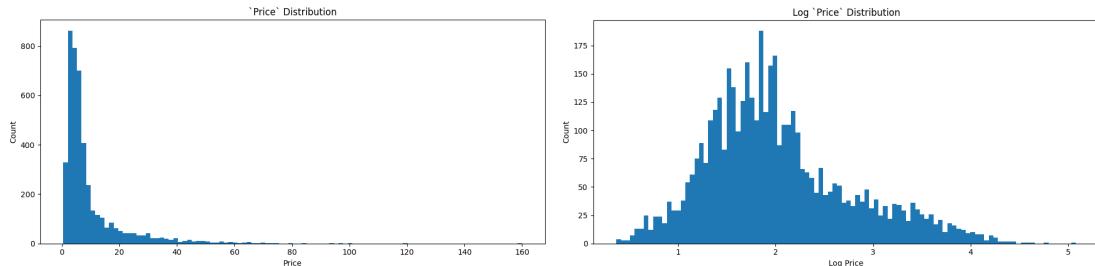


Figure 2: Imputation of Power using its correlation to Engine

Non-Linear Transformation on Price The target feature Price was heavily skewed. Applying logarithmic transformation relieved the skewness and led to 0.0150 drop of MAPE cross validation score, which indicate that the transformation was very effective.



(a) Price distribution before transformation (b) Price distribution after transformation

Figure 3: Price distribution of before and after logarithmic transformation

Adopted Pipeline As a result of feature-by-feature engineering, the adopted pipeline consisted of extracting Brand and Model, target-encoding Brand, Model, and Name, transforming Year to Age, clipping unrealistic Kilometers_Driven, grouping rare Fuel_Type, clipping unrealistic Mileage, imputing power by predicting with Fuel_Type, binning Seats, log transforming Price, category encoding, and log transforming Price. Evaluated with cross-validation, MAPE score decreased by 0.0229 compared to the baseline, which is a significant gain of performance. Details of this pipeline is included in the Table 4

2.2.3 Interaction Features

Building on top of independently engineered features, several interaction features were attempted.

Data-Driven Approach By calculating the correlation plot on the residuals of the model predictions, Engine, Price, Power, Brand, Model, and Name was found to have high correlation in the residuals. Starting with these candidates, I incrementally introduced interaction features, and some of them yielded mild performance boost.

Table 3: Interaction Feature Engineering

Transformer	Transformation Process	MAPE change	Adopted
BrandEngineInteraction	Create Brand × Engine feature	+ 0.0004	No
BrandPowerInteraction	Create Brand × Power feature	- 0.0008	Yes
AgeNewPriceInteraction	Create Age × New_Price feature	- 0.0010	No
ModelPowerInteraction	Create Model × Power feature	- 0.0018	Yes

MAPE change in Table 3 are relative to the MAPE from previous best feature-by-feature engineering result. It measures whether the introduction of interaction features was helpful. BrandPowerInteraction and ModelPowerInteraction was adopted. AgeNewPriceInteraction was dropped because it harmed performance when mixed with other feature engineering.

Ontology Based Approach In other directions, creating interaction features strongly driven by domain knowledge was attempted. Based on common knowledge, I modeled the ontology of features on car price (Figure 4). As a result, the features were hierarchically assembled to core features: BaseValueIndex, WearIndex, and MarketIndex. However, this approach harmed the model in practice, increasing MAPE by +0.0096 compared to the model without this ontology feature.

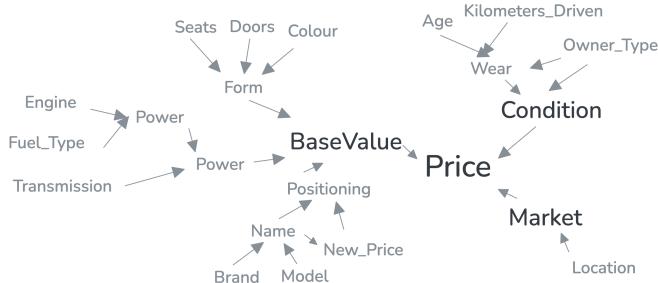


Figure 4: Ontology of features

2.3 Final Feature Engineering Pipeline

Table 4 summarizes the finalized feature engineering pipeline. MAPE change column denotes the change of cross-validation (5-fold) MAPE score when only that Transformer was applied. Cumulative MAPE change denotes the MAPE change when the pipeline upto that Transformer was applied. (As an exception, CategoryEncoder was always applied just before the model on all cumulative MAPE measures). On an untuned XGBoost model, this feature engineering pipeline yielded 0.1378 ± 0.0150 5-fold cross-validation MAPE score. The full code is attached in the Appendix A.8.

Table 4: Final feature engineering pipeline

Trasnformer	Transformation Process	MAPE change	cummulative MAPE change
BrandModelExtractor	Split Name to Brand and Model	+ 0.0299	+ 0.0299
TargetEncoder	Target Encode Brand, Model, Name, and Location	- 0.0127	- 0.0127
YearToAgeTransformer	Year -> Age = 2020 - Year	- 0.0004	- 0.0127
KilometersDrivenClipper	Clip extreme KilometersDriven	- 0.0010	- 0.0121
FuelTypeGrouper	Group CNG, LPT, and Electric	- 0.0005	- 0.0117
MileageClipper	Clip extreme Mileage	- 0.0028	- 0.0123
PowerImputer	Estimate missing Power by linear estimator w. Engine	- 0.0055	- 0.0140
SeatsBinner	Bin 2, 4-> 4 5-> 5 6, 7, 8 -> 7 9, 10 -> 9	- 0.0025	- 0.0127
NewPriceTransformer	Apply log(1+x) on New_Price	+ 0.0000	- 0.0127
BrandPowerInteraction	Create Brand × Power feature	- 0.0008	- 0.0153
ModelPowerInteraction	Create Model × Power feature	- 0.0018	- 0.0123
CategoryEncoder	Category encode all the left categorical features	*	*
PriceTransformer	Apply log(1 + x) on Price	- 0.0150	- 0.0235

3 Model Comparison and Hyperparameter Tuning

3.1 Model Comparison

XGBoost was heavily used as the base model, due to its exceptional ability to understand implicit interactions, given sufficiently clean features. CatBoost and RandomForest were evaluated on top of the same preprocessing and feature engineering, and XGBoost achieved the lowest MAPE score and standard deviation of MAPE (Table 5). Therefore, XGBoost was the most performant and stable, compared to CatBoost and RandomForest.

3.2 Hyperparameter Tuning

As a final step, I performed grid serach on the hyperparameters of XGBoost. Table 6 shows some of the best configurations for XGBoostRegressor. The code for grid search is documented in the notebook attached in Appendix A.5

Table 6: Hyperparameter Tuning

Table 5: Model comparison

Model	MAPE	MAPE Standard Deviation
XGBoost	0.1372	0.0116
CatBoost	0.1396	0.0164
Random Forest	0.1527	0.0146

Config	max_depth	reg_lambda	subsample	learning_rate	n_estimators	MAPE
1	4	2	0.9	0.10	800	0.1266
2	4	1	0.7	0.03	1500	0.1271
3	4	2	0.9	0.10	1500	0.1271
4	4	2	0.7	0.10	800	0.1272
5	4	2	0.7	0.10	1500	0.1275

4 Final Result



Figure 5: Kaggle competition final result

References

- [1] S. Kim. (2025) GIST-MLDL25f-HW3. [Online]. Available: <https://www.kaggle.com/competitions/gist-mldl-25f-hw3>

A Apendix

A.1 Source Code

The source code for this project can be found at Github <https://github.com/ParkJumyung/MLDL-HW3>. The notebooks and codes used for this project are appended to this report as a reference.

A.2 Preprocessing Notebook (*preprocessing.ipynb*)

preprocessing

November 17, 2025

0.1 Preprocessing

```
[1]: %load_ext autoreload  
%autoreload 2
```

```
[2]: from pathlib import Path  
import zipfile  
import os  
from enum import Enum  
  
import pandas as pd  
import kaggle
```

```
[3]: def download(competition: str, dir: Path | str) -> tuple[Path, Path]:  
    """  
    Downloads dataset from kaggle competition.  
  
    Args:  
        competition (str): Kaggle competition to download dataset from.  
        dir (Path / str): Path to download the dataset at.  
  
    Returns:  
        (Path, Path): Tuple of training dataset and test dataset paths.  
    """  
    if isinstance(dir, str):  
        dir = Path(dir)  
    if not dir.exists():  
        dir.mkdir(parents=True)  
  
    kaggle.api.authenticate()  
    kaggle.api.competition_download_files(competition, dir)  
    zip_file_path = Path(dir, competition).with_suffix(".zip")  
    with zipfile.ZipFile(zip_file_path, "r") as zip_ref:  
        zip_ref.extractall(dir)  
    os.remove(zip_file_path)  
  
    return (dir / "train.csv", dir / "test.csv")
```

```
[4]: train_data_path, test_data_path = download("gist-mldl-25f-hw3", "../dataset")
```

```
[5]: df_train = pd.read_csv(train_data_path)
df_test = pd.read_csv(test_data_path)
```

```
[6]: df_train
```

```
[6]:
```

ID	Name	Location	Year	Kilometers_Driven	\
0 G4XLU0	Tata Indigo	Coimbatore	2013	59138	
1 CRSHOS	Toyota Corolla	Kochi	2013	81504	
2 FUJ4X1	Ford Ikon	Hyderabad	2007	92000	
3 QMVK6E	Hyundai i20	Kolkata	2012	33249	
4 4SWHFC	Honda City	Bangalore	2011	65000	
...
4465 TR7SLB	Mahindra XUV500	Kochi	2016	51884	
4466 QB41QE	Honda Jazz	Kolkata	2016	27210	
4467 ODG8N7	Land Rover Range	Pune	2015	52000	
4468 EV2ZBX	Maruti Alto	Delhi	2013	56000	
4469 J2RCU8	Mercedes-Benz GL-Class	Bangalore	2014	52000	

Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	\
0 Diesel	Manual	First	17.0 kmpl	1405 CC	70 bhp	
1 Diesel	Manual	First	21.43 kmpl	1364 CC	87.2 bhp	
2 Petrol	Manual	First	13.8 kmpl	1299 CC	70 bhp	
3 Diesel	Manual	First	21.27 kmpl	1396 CC	88.76 bhp	
4 Petrol	Manual	First	17.0 kmpl	1497 CC	118 bhp	
...
4465 Diesel	Manual	First	16.0 kmpl	2179 CC	140 bhp	
4466 Diesel	Manual	First	27.3 kmpl	1498 CC	98.6 bhp	
4467 Diesel	Automatic	First	12.7 kmpl	2179 CC	187.7 bhp	
4468 Petrol	Manual	First	24.7 kmpl	796 CC	47.3 bhp	
4469 Diesel	Automatic	First	12.0 kmpl	2987 CC	224 bhp	

Colour	Seats	No. of Doors	New_Price	Price
0 Others	5	4	\N	2.58
1 Others	5	4	\N	6.53
2 Others	5	4	\N	1.25
3 Black/Silver	5	4	\N	3.25
4 White	5	4	\N	5.20
...
4465 White	7	5	\N	12.46
4466 Others	5	4	\N	5.85
4467 White	5	4	\N	39.75
4468 Others	5	4	\N	2.10
4469 Black/Silver	7	5	\N	49.00

[4470 rows x 16 columns]

```
[7]: df = pd.concat([df_train, df_test])
df
```

	ID	Name	Location	Year	Kilometers_Driven	Fuel_Type	\
0	G4XLU0	Tata Indigo	Coimbatore	2013	59138	Diesel	
1	CRSHOS	Toyota Corolla	Kochi	2013	81504	Diesel	
2	FUJ4X1	Ford Ikon	Hyderabad	2007	92000	Petrol	
3	QMVK6E	Hyundai i20	Kolkata	2012	33249	Diesel	
4	4SWHFC	Honda City	Bangalore	2011	65000	Petrol	
...	\
1486	CWRWOT	Tata Safari	Bangalore	2011	80000	Diesel	
1487	Q7Z939	Volkswagen Passat	Kolkata	2011	42500	Diesel	
1488	73KOPC	Audi A4	Bangalore	2014	37600	Diesel	
1489	XEBBLO	Mahindra Scorpio	Bangalore	2011	73000	Diesel	
1490	LOLVST	Hyundai i20	Coimbatore	2017	14618	Petrol	
	Transmission	Owner_Type	Mileage	Engine	Power	Colour	\
0	Manual	First	17.0 kmpl	1405 CC	70 bhp	Others	
1	Manual	First	21.43 kmpl	1364 CC	87.2 bhp	Others	
2	Manual	First	13.8 kmpl	1299 CC	70 bhp	Others	
3	Manual	First	21.27 kmpl	1396 CC	88.76 bhp	Black/Silver	
4	Manual	First	17.0 kmpl	1497 CC	118 bhp	White	
...	\
1486	Manual	First	13.93 kmpl	2179 CC	138.03 bhp	Others	
1487	Automatic	First	18.33 kmpl	1968 CC	167.7 bhp	Black/Silver	
1488	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	Black/Silver	
1489	Manual	First	12.05 kmpl	2179 CC	120 bhp	Others	
1490	Manual	First	18.6 kmpl	1197 CC	81.83 bhp	Black/Silver	
	Seats	No. of Doors	New_Price	Price			
0	5	4	\N	2.58			
1	5	4	\N	6.53			
2	5	4	\N	1.25			
3	5	4	\N	3.25			
4	5	4	\N	5.20			
...			
1486	7	5	\N	NaN			
1487	5	4	\N	NaN			
1488	5	4	\N	NaN			
1489	8	5	\N	NaN			
1490	5	4	\N	NaN			

[5961 rows x 16 columns]

1. ID

Unique identifier for each car listing

```
[8]: df.ID.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, 0 to 1490
Series name: ID
Non-Null Count Dtype
-----
5961 non-null object
dtypes: object(1)
memory usage: 93.1+ KB
```

```
[9]: len(df.ID.unique()) == len(df.ID)
```

```
[9]: True
```

ID feature has no missing value and all unique.

```
[10]: df.set_index("ID", inplace=True)
df
```

```
[10]:
```

ID	Name	Location	Year	Kilometers_Driven	Fuel_Type	\	
G4XLU0	Tata Indigo	Coimbatore	2013	59138	Diesel		
CRSHOS	Toyota Corolla	Kochi	2013	81504	Diesel		
FUJ4X1	Ford Ikon	Hyderabad	2007	92000	Petrol		
QMVK6E	Hyundai i20	Kolkata	2012	33249	Diesel		
4SWHFC	Honda City	Bangalore	2011	65000	Petrol		
...		
CWRWOT	Tata Safari	Bangalore	2011	80000	Diesel		
Q7Z939	Volkswagen Passat	Kolkata	2011	42500	Diesel		
73KOPC	Audi A4	Bangalore	2014	37600	Diesel		
XEBBL0	Mahindra Scorpio	Bangalore	2011	73000	Diesel		
LOLVST	Hyundai i20	Coimbatore	2017	14618	Petrol		
ID	Transmission	Owner_Type	Mileage	Engine	Power	Colour	\
G4XLU0	Manual	First	17.0 kmpl	1405 CC	70 bhp	Others	
CRSHOS	Manual	First	21.43 kmpl	1364 CC	87.2 bhp	Others	
FUJ4X1	Manual	First	13.8 kmpl	1299 CC	70 bhp	Others	
QMVK6E	Manual	First	21.27 kmpl	1396 CC	88.76 bhp	Black/Silver	
4SWHFC	Manual	First	17.0 kmpl	1497 CC	118 bhp	White	
...	
CWRWOT	Manual	First	13.93 kmpl	2179 CC	138.03 bhp	Others	
Q7Z939	Automatic	First	18.33 kmpl	1968 CC	167.7 bhp	Black/Silver	
73KOPC	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	Black/Silver	
XEBBL0	Manual	First	12.05 kmpl	2179 CC	120 bhp	Others	
LOLVST	Manual	First	18.6 kmpl	1197 CC	81.83 bhp	Black/Silver	

	Seats	No. of Doors	New_Price	Price
ID				
G4XLU0	5	4	\N	2.58
CRSHOS	5	4	\N	6.53
FUJ4X1	5	4	\N	1.25
QMVK6E	5	4	\N	3.25
4SWHFC	5	4	\N	5.20
...
CWRWOT	7	5	\N	NaN
Q7Z939	5	4	\N	NaN
73KOPC	5	4	\N	NaN
XEBBLO	8	5	\N	NaN
LOLVST	5	4	\N	NaN

[5961 rows x 15 columns]

2. Name

The brand and model name of the car (e.g. Hyundai i20, Honda City)

[11]: df.Name.info()

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Name
Non-Null Count Dtype
-----
5961 non-null    object
dtypes: object(1)
memory usage: 93.1+ KB
```

[12]: df.Name.unique()

```
[12]: array(['Tata Indigo', 'Toyota Corolla', 'Ford Ikon', 'Hyundai i20',
       'Honda City', 'Ford Ecosport', 'Hyundai Grand', 'Maruti Wagon',
       'Mercedes-Benz GLA', 'Jaguar XF', 'Porsche Cayenne', 'BMW 3',
       'Mercedes-Benz New', 'Tata Manza', 'Fiat Linea', 'Maruti Swift',
       'Mercedes-Benz GLE', 'BMW 5', 'Ford Fiesta', 'Honda Accord',
       'Maruti Alto', 'Mahindra XUV500', 'Fiat Petra', 'Skoda Laura',
       'Maruti Baleno', 'Jeep Compass', 'BMW X1', 'Hyundai EON',
       'Ford Figo', 'Hyundai i10', 'Toyota Innova', 'Renault Duster',
       'Skoda Superb', 'Toyota Etios', 'Hyundai Verna', 'Honda WRV',
       'Mahindra Scorpio', 'Maruti Esteem', 'Nissan Sunny',
       'Nissan Terrano', 'Audi Q3', 'Ford EcoSport', 'BMW Z4',
       'Maruti Dzire', 'BMW X5', 'Audi Q7', 'Honda Amaze',
       'Mercedes-Benz E-Class', 'Volkswagen Polo', 'Tata Indica',
       'Chevrolet Cruze', 'Maruti Ertiga', 'Chevrolet Spark',
       'Mercedes-Benz A', 'Maruti Eeco', 'Honda Brio', 'Ford Endeavour'],
```

```
'Mercedes-Benz M-Class', 'Hyundai Creta', 'Volkswagen Vento',
'Hyundai Xcent', 'Audi A7', 'Mercedes-Benz CLA', 'Skoda Octavia',
'Chevrolet Captiva', 'Tata New', 'Force One', 'Honda Jazz',
'Mahindra Bolero', 'BMW X3', 'Jaguar F', 'Skoda Fabia',
'Mitsubishi Cedia', 'Tata Xenon', 'Maruti Ritz', 'BMW 7',
'Mahindra Xylo', 'Maruti Vitara', 'Maruti Zen', 'Toyota Fortuner',
'Mahindra Renault', 'Hyundai Elantra', 'Fiat Siena',
'Honda Mobilio', 'Chevrolet Beat', 'Mahindra Ssangyong',
'Tata Safari', 'Renault KWID', 'Mercedes-Benz GLS', 'Honda Civic',
'Volkswagen Ameo', 'Maruti 800', 'Audi A4', 'Chevrolet Enjoy',
'Honda CR-V', 'Hyundai Accent', 'Maruti Grand', 'Skoda Rapid',
'Tata Nano', 'Mercedes-Benz B', 'Audi Q5', 'Honda BRV',
'Land Rover Range', 'Mahindra KUV', 'Volkswagen Passat',
'Maruti Ignis', 'Renault Captur', 'Datsun redi-GO', 'Jaguar XJ',
'Maruti SX4', 'Mercedes-Benz GL-Class', 'Maruti Ciaz',
'Maruti Celerio', 'Mahindra XUV300', 'Mahindra TUV',
'Hyundai Santro', 'Tata Zest', 'Mercedes-Benz GLC',
'Volkswagen Jetta', 'Datsun Redi', 'Chevrolet Optra',
'Maruti Omni', 'Maruti A-Star', 'Audi A6', 'Maruti S',
'Mini Cooper', 'Mahindra Logan', 'Chevrolet Aveo',
'Mercedes-Benz S', 'Hyundai Santa', 'Mercedes-Benz C-Class',
'Honda BR-V', 'Tata Sumo', 'Smart Fortwo', 'Fiat Grande',
'Mitsubishi Montero', 'Chevrolet Sail', 'Audi RS5',
'Porsche Panamera', 'Land Rover Discovery', 'Mahindra Thar',
'Tata Nexon', 'Mahindra Quanto', 'Tata Tiago', 'Mitsubishi Pajero',
'Isuzu MUX', 'Land Rover Freelander', 'Ford Fusion', 'Mahindra E',
'Skoda Yeti', 'Hyundai Tucson', 'Mahindra Verito', 'Datsun GO',
'Nissan Micra', 'Renault Fluence', 'Renault Pulse',
'Mercedes-Benz R-Class', 'Volvo XC60', 'BMW 6', 'Tata Tigor',
'BMW X6', 'Volvo S60', 'Mercedes-Benz S-Class', 'Toyota Camry',
'Nissan X-Trail', 'Ford Aspire', 'Ford Freestyle', 'Tata Bolt',
'ISUZU D-MAX', 'Toyota Qualis', 'Porsche Boxster', 'Hyundai Elite',
'Hyundai Getz', 'Fiat Punto', 'Hyundai Sonata', 'Porsche Cayman',
'Jaguar XE', 'Audi A8', 'Maruti S-Cross', 'Fiat Avventura',
'Volkswagen CrossPolo', 'Toyota Prius', 'Volvo V40',
'Renault Scala', 'Bentley Continental', 'Tata Hexa', 'Audi A3',
'Mahindra Jeep', 'Mitsubishi Lancer', 'Mahindra NuvoSport',
'Renault Koleos', 'BMW 1', 'Volvo S80', 'Mini Clubman',
'Mercedes-Benz SLK-Class', 'Toyota Platinum', 'Mini Countryman',
'Volkswagen Beetle', 'Nissan Evalia', 'Mercedes-Benz SLC',
'Audi TT', 'Nissan Teana', 'Mercedes-Benz CLS-Class',
'Lamborghini Gallardo', 'Honda WR-V', 'Mitsubishi Outlander',
'Volvo XC90', 'Mercedes-Benz SL-Class', 'Ford Classic',
'Volkswagen Tiguan', 'Ford Mustang', 'Maruti 1000'], dtype=object)
```

Name encodes both the brand and model. Since the brand itself is considered to be a critical feature, separate Name into Brand and Model. Note that Brand and Model has a hierarchical relationship. Not all of the cartesian product of Brand and Model is valid.

Naive string splitting won't work due to edge cases like Land Rover Range and Mahindra Ssangyong.

```
[13]: class Brand(Enum):
    """
    Enum of recognized brands. All the values are in title-case. Transform to
    ↵title-case before comparing.
    """

    Audi = "Audi"
    Bentley = "Bentley"
    BMW = "Bmw"
    Chevrolet = "Chevrolet"
    Datsun = "Datsun"
    Fiat = "Fiat"
    Force = "Force"
    Ford = "Ford"
    Honda = "Honda"
    Hyundai = "Hyundai"
    Isuzu = "Isuzu"
    Jaguar = "Jaguar"
    Jeep = "Jeep"
    Lamborghini = "Lamborghini"
    Land_Rover = "Land Rover"
    Mahindra = "Mahindra"
    Maruti = "Maruti"
    Mercedes_Benz = "Mercedes-Benz"
    Mini = "Mini"
    Mitsubishi = "Mitsubishi"
    Nissan = "Nissan"
    Porsche = "Porsche"
    Renault = "Renault"
    Skoda = "Skoda"
    Smart = "Smart"
    Tata = "Tata"
    Toyota = "Toyota"
    Volkswagen = "Volkswagen"
    Volvo = "Volvo"
```

```
[14]: for name in df.Name.unique():
    brands = tuple(brand.value for brand in set(Brand))
    if not name.title().startswith(brands):
        print(f"Could not match any brand in the name: {name}")
        break
    else:
        print("Every brand name recognized!")
```

Every brand name recognized!

```
[15]: def split_brand_and_model_from_name(series: pd.Series) -> tuple[pd.Series, pd.Series]:
    """
    From a given series of car names, it splits the name into brand and model.

    Args:
        series (Series): A series of names to be processed.

    Returns:
        tuple[Series, Series]: A tuple of Brand series and Model series.
    """
    names = series.str.title()
    brands = pd.Series([None] * len(names), index=names.index, dtype=object)
    models = pd.Series([None] * len(names), index=names.index, dtype=object)

    for brand in Brand:
        condition = names.str.startswith(brand.value, na=False) & brands.isna()

        if condition.any():
            brands.loc[condition] = brand.name
            matched_names = names.loc[condition]
            residuals = matched_names.str[len(brand.value):].str.strip()
            models.loc[condition] = residuals.where(residuals != "", None)

    brands = brands.rename("Brand").astype("category")
    models = models.rename("Model").astype("category")

    return brands, models
```

```
[16]: df["Brand"], df["Model"] = split_brand_and_model_from_name(df.Name)
df
```

ID	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Colour
G4XLU0	Tata Indigo	Coimbatore	2013	59138	Diesel						
CRSHOS	Toyota Corolla	Kochi	2013	81504	Diesel						
FUJ4X1	Ford Ikon	Hyderabad	2007	92000	Petrol						
QMVK6E	Hyundai i20	Kolkata	2012	33249	Diesel						
4SWHFC	Honda City	Bangalore	2011	65000	Petrol						
...						
CWRWOT	Tata Safari	Bangalore	2011	80000	Diesel						
Q7Z939	Volkswagen Passat	Kolkata	2011	42500	Diesel						
73KOPC	Audi A4	Bangalore	2014	37600	Diesel						
XEBBL0	Mahindra Scorpio	Bangalore	2011	73000	Diesel						
LOLVST	Hyundai i20	Coimbatore	2017	14618	Petrol						

ID							
G4XLU0	Manual	First	17.0 kmpl	1405 CC	70 bhp	Others	
CRSHOS	Manual	First	21.43 kmpl	1364 CC	87.2 bhp	Others	
FUJ4X1	Manual	First	13.8 kmpl	1299 CC	70 bhp	Others	
QMVK6E	Manual	First	21.27 kmpl	1396 CC	88.76 bhp	Black/Silver	
4SWHFC	Manual	First	17.0 kmpl	1497 CC	118 bhp	White	
...	
CWRWOT	Manual	First	13.93 kmpl	2179 CC	138.03 bhp	Others	
Q7Z939	Automatic	First	18.33 kmpl	1968 CC	167.7 bhp	Black/Silver	
73KOPC	Automatic	Second	16.55 kmpl	1968 CC	147.51 bhp	Black/Silver	
XEBBL0	Manual	First	12.05 kmpl	2179 CC	120 bhp	Others	
LOLVST	Manual	First	18.6 kmpl	1197 CC	81.83 bhp	Black/Silver	

ID	Seats	No. of Doors	New_Price	Price	Brand	Model
G4XLU0	5	4	\N	2.58	Tata	Indigo
CRSHOS	5	4	\N	6.53	Toyota	Corolla
FUJ4X1	5	4	\N	1.25	Ford	Ikon
QMVK6E	5	4	\N	3.25	Hyundai	I20
4SWHFC	5	4	\N	5.20	Honda	City
...
CWRWOT	7	5	\N	NaN	Tata	Safari
Q7Z939	5	4	\N	NaN	Volkswagen	Passat
73KOPC	5	4	\N	NaN	Audi	A4
XEBBL0	8	5	\N	NaN	Mahindra	Scorpio
LOLVST	5	4	\N	NaN	Hyundai	I20

[5961 rows x 17 columns]

```
[17]: groupby_brand = df.groupby("Brand", observed=True)
brand_histogram = groupby_brand.nunique()[["Model"]].sort_values(ascending=False)
brand_histogram
```

Brand	
Maruti	22
Mercedes_Benz	19
Mahindra	16
Hyundai	15
Tata	14
Honda	12
Ford	10
BMW	10
Audi	10
Toyota	8
Chevrolet	8
Volkswagen	8
Renault	7

```
Fiat          6
Skoda         6
Nissan        6
Mitsubishi    5
Volvo          5
Porsche        4
Jaguar          4
Mini            3
Datsun          3
Land_Rover     3
Isuzu           2
Lamborghini    1
Jeep             1
Smart            1
Force            1
Bentley          1
Name: Model, dtype: int64
```

```
[18]: with pd.option_context(
    "display.max_rows",
    None,
):
    display(df[["Brand", "Model"]].groupby(["Brand", "Model"], observed=True) .
    ↪count())
```

```
Empty DataFrame
Columns: []
```

```
Index: [(Audi, A3), (Audi, A4), (Audi, A6), (Audi, A7), (Audi, A8), (Audi, Q3),  
        (Audi, Q5), (Audi, Q7), (Audi, Rs5), (Audi, Tt), (BMW, 1), (BMW, 3), (BMW, 5),  
        (BMW, 6), (BMW, 7), (BMW, X1), (BMW, X3), (BMW, X5), (BMW, X6), (BMW, Z4),  
        (Bentley, Continental), (Chevrolet, Aveo), (Chevrolet, Beat), (Chevrolet,  
        Captiva), (Chevrolet, Cruze), (Chevrolet, Enjoy), (Chevrolet, Optra),  
        (Chevrolet, Sail), (Chevrolet, Spark), (Datsun, Go), (Datsun, Redi), (Datsun,  
        Redi-Go), (Fiat, Avventura), (Fiat, Grande), (Fiat, Linea), (Fiat, Petra),  
        (Fiat, Punto), (Fiat, Siena), (Force, One), (Ford, Aspire), (Ford, Classic),  
        (Ford, Ecosport), (Ford, Endeavour), (Ford, Fiesta), (Ford, Figo), (Ford,  
        Freestyle), (Ford, Fusion), (Ford, Ikon), (Ford, Mustang), (Honda, Accord),  
        (Honda, Amaze), (Honda, Br-V), (Honda, Brio), (Honda, Brv), (Honda, City),  
        (Honda, Civic), (Honda, Cr-V), (Honda, Jazz), (Honda, Mobilio), (Honda, Wr-V),  
        (Honda, Wrv), (Hyundai, Accent), (Hyundai, Creta), (Hyundai, Elantra),  
        (Hyundai, Elite), (Hyundai, Eon), (Hyundai, Getz), (Hyundai, Grand), (Hyundai,  
        I10), (Hyundai, I20), (Hyundai, Santa), (Hyundai, Santro), (Hyundai, Sonata),  
        (Hyundai, Tucson), (Hyundai, Verna), (Hyundai, Xcent), (Isuzu, D-Max), (Isuzu,  
        Mux), (Jaguar, F), (Jaguar, Xe), (Jaguar, Xf), (Jaguar, Xj), (Jeep, Compass),  
        (Lamborghini, Gallardo), (Land_Rover, Discovery), (Land_Rover, Freelander),  
        (Land_Rover, Range), (Mahindra, Bolero), (Mahindra, E), (Mahindra, Jeep),  
        (Mahindra, Kuv), (Mahindra, Logan), (Mahindra, Nuvosport), (Mahindra, Quanto),  
        (Mahindra, Renault), (Mahindra, Scorpio), (Mahindra, Ssangyong), (Mahindra,  
        Thar), (Mahindra, Tuv), (Mahindra, Verito), ...]
```

```
[19]: df.drop(columns=["Name"], inplace=True)
```

3. Location

The city where the car is being sold

```
[20]: df.Location.info()
```

```
<class 'pandas.core.series.Series'>  
Index: 5961 entries, G4XLU0 to LOLVST  
Series name: Location  
Non-Null Count Dtype  
-----  
5961 non-null object  
dtypes: object(1)  
memory usage: 222.2+ KB
```

```
[21]: df.Location.unique()
```

```
[21]: array(['Coimbatore', 'Kochi', 'Hyderabad', 'Kolkata', 'Bangalore',  
           'Delhi', 'Pune', 'Chennai', 'Mumbai', 'Ahmedabad', 'Jaipur', '\\N'],  
           dtype=object)
```

These seem to be locations in India. Some missing values are denoted with \\N.

```
[22]: df.Location = df.Location.replace("\\\\N", None).astype("category")
df.Location.unique()
```

```
[22]: ['Coimbatore', 'Kochi', 'Hyderabad', 'Kolkata', 'Bangalore', ..., 'Chennai',
'Mumbai', 'Ahmedabad', 'Jaipur', NaN]
Length: 12
Categories (11, object): ['Ahmedabad', 'Bangalore', 'Chennai', 'Coimbatore',
..., 'Kochi', 'Kolkata', 'Mumbai', 'Pune']
```

```
[23]: df.Location.value_counts()
```

```
[23]: Location
Mumbai      781
Hyderabad   739
Kochi       646
Coimbatore  630
Pune        611
Delhi        549
Kolkata     526
Chennai      489
Jaipur       406
Bangalore    351
Ahmedabad   222
Name: count, dtype: int64
```

There seems to be no location with extremely small sample size.

4. Year

Manufacturing year of the vehicle

```
[24]: df.Year.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Year
Non-Null Count   Dtype  
----- 
5961 non-null    object 
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[25]: df.Year.unique()
```

```
[25]: array(['2013', '2007', '2012', '2011', '2014', '2016', '2019', '2015',
'2008', '2010', '2017', '2005', '2009', '2018', '2004', '2006',
'2001', '1999', '2002', '2003', '2000', '\\N', '1998', 2012, 2008,
2010, 2017, 2014, 2011, 2015, 2018, 2016, 2009, 2013, 2004, 2007,
2019, 1998, 2005, 2000, 2006, 2003, 2002, 2001], dtype=object)
```

Similarly, missing values are represented with \\N.

```
[26]: df.Year = df.Year.replace("\\N", None)
df.Year.unique()
```

```
[26]: array(['2013', '2007', '2012', '2011', '2014', '2016', '2019', '2015',
       '2008', '2010', '2017', '2005', '2009', '2018', '2004', '2006',
       '2001', '1999', '2002', '2003', '2000', None, '1998', 2012, 2008,
      2010, 2017, 2014, 2011, 2015, 2018, 2016, 2009, 2013, 2004, 2007,
      2019, 1998, 2005, 2000, 2006, 2003, 2002, 2001], dtype=object)
```

```
[27]: pd.to_numeric(df.Year, errors="coerce").astype("Int64").value_counts().
      ↪sort_index()
```

```
[27]: Year
1998      4
1999      2
2000      4
2001      7
2002     14
2003     13
2004     28
2005     55
2006     75
2007    123
2008    170
2009    196
2010    338
2011    461
2012    573
2013    642
2014    793
2015    736
2016    740
2017    586
2018    298
2019    101
Name: count, dtype: Int64
```

There are some years with significantly small sample size.

Setting the year 2020 as the current year, converting year to Age might be more intuitive value relevant to New_Price.

```
[28]: current_year = 2020
df["Age"] = current_year - pd.to_numeric(df.Year, errors="coerce").
      ↪astype("Int64")
```

```
df.Age.value_counts(sort=False).sort_index()
```

[28]: Age

```
1      101
2      298
3      586
4      740
5      736
6      793
7      642
8      573
9      461
10     338
11     196
12     170
13     123
14     75
15     55
16     28
17     13
18     14
19      7
20      4
21      2
22      4
Name: count, dtype: Int64
```

5. Kilometers Driven

Total distance the car has been driven (in kilometers)

[29]: df.Kilometers_Driven.info()

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Kilometers_Driven
Non-Null Count    Dtype
-----
5961 non-null    object
dtypes: object(1)
memory usage: 222.2+ KB
```

[30]: df.Kilometers_Driven = df.Kilometers_Driven.replace("\\\n", None)
df[df.Kilometers_Driven.isna()]

[30]: Location Year Kilometers_Driven Fuel_Type Transmission Owner_Type \
ID
VA5F28 NaN 2017 None Petrol Manual First

BVPJHJ	NaN	2013	None	Diesel	Automatic	Second
5ZGUKG	NaN	2018	None	Diesel	Manual	First
LWTYCE	NaN	2013	None	Diesel	Automatic	First
HZTZU8	NaN	2014	None	Petrol	Manual	Second
CZ59WU	NaN	2010	None	Diesel	Automatic	First
EGWEU4	NaN	2009	None	Petrol	Manual	Second
MGGIOB	NaN	2012	None	Petrol	Manual	First
ID	Mileage	Engine	Power	Colour	Seats No. of Doors	New_Price \
VA5F28	18.15 kmpl	1198 CC	82 bhp	Others	5	4 \N
BVPJHJ	11.18 kmpl	2696 CC	184 bhp	Others	7	5 \N
5ZGUKG	24.3 kmpl	1248 CC	88.5 bhp	White	5	4 11.12 Lakh
LWTYCE	11.74 kmpl	2987 CC	254.8 bhp	Others	5	4 \N
HZTZU8	16.09 kmpl	1598 CC	103.5 bhp	White	5	4 12.33 Lakh
CZ59WU	12.4 kmpl	2698 CC	179.5 bhp	Others	5	4 \N
EGWEU4	14.0 kmpl	1061 CC	64 bhp	White	5	4 \N
MGGIOB	14.53 kmpl	1798 CC	138.1 bhp	White	5	4 \N
ID	Price	Brand	Model	Age		
VA5F28	4.85	Mahindra	Kuv	3		
BVPJHJ	12.50	Mahindra	Ssangyong	7		
5ZGUKG	8.63	Maruti	Vitara	2		
LWTYCE	28.00	Mercedes_Benz	M-Class	7		
HZTZU8	6.98	Volkswagen	Vento	6		
CZ59WU	NaN	Audi	A6	10		
EGWEU4	NaN	Maruti	Wagon	11		
MGGIOB	NaN	Toyota	Corolla	8		

```
[31]: df.Kilometers_Driven = pd.to_numeric(df.Kilometers_Driven)
df.Kilometers_Driven = df.Kilometers_Driven.astype("Int64")
df.Kilometers_Driven
```

```
[31]: ID
G4XLU0      59138
CRSHOS      81504
FUJ4X1       92000
QMVK6E      33249
4SWHFC      65000
...
CWRWOT      80000
Q7Z939      42500
73KOPC      37600
XEBBL0      73000
LOLVST      14618
Name: Kilometers_Driven, Length: 5961, dtype: Int64
```

6. Fuel_Type

Type of fuel the car uses (Pertrol, Diesel, CNG, LPG, Electric)

[32]: df.Fuel_Type.info()

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Fuel_Type
Non-Null Count Dtype
-----
5961 non-null object
dtypes: object(1)
memory usage: 222.2+ KB
```

[33]: df.Fuel_Type.unique()

[33]: array(['Diesel', 'Petrol', 'CNG', 'LPG', 'Electric'], dtype=object)

[34]: df.Fuel_Type = df.Fuel_Type.astype("category")
df.Fuel_Type.cat.categories

[34]: Index(['CNG', 'Diesel', 'Electric', 'LPG', 'Petrol'], dtype='object')

[35]: df.Fuel_Type.value_counts()

```
[35]: Fuel_Type
Diesel      3188
Petrol      2705
CNG         56
LPG          10
Electric      2
Name: count, dtype: int64
```

7. Transmission

Type of transmission system (Manual or Automatic)

[36]: df.Transmission.info()

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Transmission
Non-Null Count Dtype
-----
5961 non-null object
dtypes: object(1)
memory usage: 222.2+ KB
```

[37]: df.Transmission.unique()

```
[37]: array(['Manual', 'Automatic', '\\N'], dtype=object)
```

```
[38]: df.Transmission.value_counts()
```

[38]: Transmission

	Manual	Automatic	\\N
Name: count, dtype: int64	4225	1709	27

```
[39]: df.Transmission = df.Transmission.replace("\\N", None)
df[df.Transmission.isna()]
```

[39]:

ID	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
VVH3NN	Chennai	2012	65932	Diesel	None	Second
HFU1G9	Jaipur	2015	100000	Diesel	None	\\N
43PACK	Coimbatore	2013	40670	Diesel	None	Second
JRH5YV	Chennai	2011	46000	Petrol	None	First
FIU4TU	Hyderabad	2013	40000	Diesel	None	\\N
6FJFYS	Hyderabad	2012	75000	LPG	None	First
9FLMYR	Pune	2015	41000	Diesel	None	First
BIV06Q	Coimbatore	2015	70602	Diesel	None	\\N
GWE95I	Mumbai	2010	72000	CNG	None	First
S149E7	Delhi	2011	68000	Diesel	None	\\N
ABRKHB	Coimbatore	2007	66800	Petrol	None	\\N
HIJ508	Chennai	2012	87000	Diesel	None	First
9Z2LPT	Mumbai	2009	102002	Diesel	None	\\N
VRWP96	Mumbai	2016	36000	Diesel	None	First
MTRG04	Bangalore	2015	67600	Petrol	None	\\N
N2LG5N	Kochi	2018	25692	Petrol	None	First
7NKZMQ	Pune	2016	37208	Diesel	None	\\N
3WVXX0	Coimbatore	2011	45004	Diesel	None	\\N
HCYSXM	Hyderabad	2009	53000	Petrol	None	\\N
6FH9L9	Delhi	2014	27365	Diesel	None	\\N
QWA2Y3	Mumbai	2011	38000	Petrol	None	\\N
V871B7	Jaipur	2013	86999	Diesel	None	First
MTN68R	Mumbai	2015	33500	Petrol	None	\\N
3D7ZY1	Pune	2013	64430	Diesel	None	First
YQ2I8J	Delhi	2013	33746	Petrol	None	\\N
G7M8HP	Kolkata	2012	60000	Petrol	None	First
H37YTM	Bangalore	2012	45000	Petrol	None	Second
ID	Mileage	Engine	Power	Colour	Seats	No. of Doors
VVH3NN	22.3 kmp	1248 CC	74 bhp	White	5	4
HFU1G9	24.4 kmp	1120 CC	71 bhp	Others	5	4

43PACK	15.2	kmpl	1968	CC	140.8	bhp	White	5	4
JRH5YV	18.2	kmpl	1199	CC	88.7	bhp	White	5	4
FIU4TU	17.85	kmpl	2967	CC	300	bhp	Black/Silver	4	4
6FJFYS	21.1	km/kg	814	CC	55.2	bhp	White	5	4
9FLMYR	19.67	kmpl	1582	CC	126.2	bhp	White	5	4
BIV06Q	25.8	kmpl	1498	CC	98.6	bhp	Others	5	4
GWE95I	26.6	km/kg	998	CC	58.16	bhp	White	5	4
S149E7	19.3	kmpl	1248	CC	73.9	bhp	Black/Silver	5	4
ABRKHB	15.3	kmpl	1341	CC	83	bhp	Others	5	4
HIJ508	20.77	kmpl	1248	CC	88.76	bhp	Others	7	5
9Z2LPT	8.7	kmpl	2987	CC	224.34	bhp	Others	5	4
VRWP96	11.36	kmpl	2755	CC	171.5	bhp	White	8	5
MTRG04	23.1	kmpl	998	CC	67.04	bhp	Black/Silver	5	4
N2LG5N	21.56	kmpl	1462	CC	103.25	bhp	Others	5	4
7NKZMQ	24.3	kmpl	1248	CC	88.5	bhp	White	5	4
3WVXX0	12.8	kmpl	2494	CC	102	bhp	Others	7	4
HCYSXM	0.0	kmpl	3597	CC	262.6	bhp	White	5	4
6FH9L9	28.4	kmpl	1248	CC	74	bhp	Black/Silver	5	4
QWA2Y3	16.09	kmpl	1598	CC	103.5	bhp	Black/Silver	5	4
V871B7	23.08	kmpl	1461	CC	63.1	bhp	White	5	4
MTN68R	19.16	kmpl	2494	CC	158.2	bhp	Others	5	4
3D7ZY1	20.54	kmpl	1598	CC	103.6	bhp	White	5	4
YQ2I8J	18.5	kmpl	1198	CC	86.8	bhp	White	5	4
G7M8HP	16.8	kmpl	1497	CC	116.3	bhp	White	5	4
H37YTM	19.4	kmpl	1198	CC	86.8	bhp	White	5	4

ID	New_Price	Price	Brand	Model	Age
VVH3NN	\N	1.95	Tata	Indica	8
HFU1G9	\N	4.00	Hyundai	Xcent	5
43PACK	\N	17.74	Audi	A4	7
JRH5YV	8.61	Lakh	Honda	Jazz	9
FIU4TU	\N	45.00	Porsche	Panamera	7
6FJFYS	\N	2.35	Hyundai	Eon	8
9FLMYR	\N	12.50	Hyundai	Creta	5
BIV06Q	\N	4.83	Honda	Amaze	5
GWE95I	\N	1.75	Maruti	Wagon	10
S149E7	\N	2.75	Maruti	Swift	9
ABRKHB	\N	2.20	Hyundai	Getz	13
HIJ508	\N	6.00	Maruti	Ertiga	8
9Z2LPT	\N	10.75	Mercedes_Benz	M-Class	11
VRWP96	21	Lakh	Toyota	Innova	4
MTRG04	\N	4.00	Maruti	Celerio	5
N2LG5N	10.65	Lakh	Maruti	Ciaz	2
7NKZMQ	9.93	Lakh	Maruti	Vitara	4
3WVXX0	\N	9.48	Toyota	Innova	9
HCYSXM	\N	NaN	Skoda	Superb	11

6FH9L9	7.88 Lakh	NaN	Maruti	Swift	6
QWA2Y3	11.91 Lakh	NaN	Volkswagen	Vento	9
V871B7	\N	NaN	Nissan	Micra	7
MTN68R	\N	NaN	Toyota	Camry	5
3D7ZY1	\N	NaN	Volkswagen	Vento	7
YQ2I8J	6.63 Lakh	NaN	Honda	Brio	7
G7M8HP	\N	NaN	Honda	City	8
H37YTM	\N	NaN	Honda	Brio	8

```
[40]: df.Transmission = df.Transmission.astype("category")
df.Transmission.cat.categories
```

```
[40]: Index(['Automatic', 'Manual'], dtype='object')
```

8. Owner_Type

Number of previous owners

```
[41]: df.Owner_Type.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Owner_Type
Non-Null Count Dtype
-----
5961 non-null    object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[42]: df.Owner_Type.unique()
```

```
[42]: array(['First', 'Second', 'Third', '\N', 'Fourth & Above'], dtype=object)
```

```
[43]: df.Owner_Type = df.Owner_Type.replace("\N", None)
df[df.Owner_Type.isna()]
```

ID	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
2BGQXK	Delhi	2012	62000	Diesel	Manual	None
HFU1G9	Jaipur	2015	100000	Diesel	NaN	None
FIU4TU	Hyderabad	2013	40000	Diesel	NaN	None
BIV06Q	Coimbatore	2015	70602	Diesel	NaN	None
S149E7	Delhi	2011	68000	Diesel	NaN	None
ABRKHB	Coimbatore	2007	66800	Petrol	NaN	None
9Z2LPT	Mumbai	2009	102002	Diesel	NaN	None
MTRG04	Bangalore	2015	67600	Petrol	NaN	None
7NKZMQ	Pune	2016	37208	Diesel	NaN	None
3WVXX0	Coimbatore	2011	45004	Diesel	NaN	None

HCYSXM	Hyderabad	2009	53000	Petrol	NaN	None
6FH9L9	Delhi	2014	27365	Diesel	NaN	None
QWA2Y3	Mumbai	2011	38000	Petrol	NaN	None
MTN68R	Mumbai	2015	33500	Petrol	NaN	None
YQ2I8J	Delhi	2013	33746	Petrol	NaN	None

ID	Mileage	Engine	Power	Colour	Seats	No. of Doors	\
2BGQXK	22.32 kmpl	1582 CC	126.32 bhp	White	5	4	
HFU1G9	24.4 kmpl	1120 CC	71 bhp	Others	5	4	
FIU4TU	17.85 kmpl	2967 CC	300 bhp	Black/Silver	4	4	
BIV06Q	25.8 kmpl	1498 CC	98.6 bhp	Others	5	4	
S149E7	19.3 kmpl	1248 CC	73.9 bhp	Black/Silver	5	4	
ABRKHB	15.3 kmpl	1341 CC	83 bhp	Others	5	4	
9Z2LPT	8.7 kmpl	2987 CC	224.34 bhp	Others	5	4	
MTRG04	23.1 kmpl	998 CC	67.04 bhp	Black/Silver	5	4	
7NKZMQ	24.3 kmpl	1248 CC	88.5 bhp	White	5	4	
3WVXX0	12.8 kmpl	2494 CC	102 bhp	Others	7	4	
HCYSXM	0.0 kmpl	3597 CC	262.6 bhp	White	5	4	
6FH9L9	28.4 kmpl	1248 CC	74 bhp	Black/Silver	5	4	
QWA2Y3	16.09 kmpl	1598 CC	103.5 bhp	Black/Silver	5	4	
MTN68R	19.16 kmpl	2494 CC	158.2 bhp	Others	5	4	
YQ2I8J	18.5 kmpl	1198 CC	86.8 bhp	White	5	4	

ID	New_Price	Price	Brand	Model	Age
2BGQXK	\N	4.60	Hyundai	Verna	8
HFU1G9	\N	4.00	Hyundai	Xcent	5
FIU4TU	\N	45.00	Porsche	Panamera	7
BIV06Q	\N	4.83	Honda	Amaze	5
S149E7	\N	2.75	Maruti	Swift	9
ABRKHB	\N	2.20	Hyundai	Getz	13
9Z2LPT	\N	10.75	Mercedes_Benz	M-Class	11
MTRG04	\N	4.00	Maruti	Celerio	5
7NKZMQ	9.93 Lakh	7.43	Maruti	Vitara	4
3WVXX0	\N	9.48	Toyota	Innova	9
HCYSXM	\N	NaN	Skoda	Superb	11
6FH9L9	7.88 Lakh	NaN	Maruti	Swift	6
QWA2Y3	11.91 Lakh	NaN	Volkswagen	Vento	9
MTN68R	\N	NaN	Toyota	Camry	5
YQ2I8J	6.63 Lakh	NaN	Honda	Brio	7

Owner_Type is ordinal category. It should be orderable.

```
[44]: owner_type_order = {"First": 1, "Second": 2, "Third": 3, "Fourth & Above": 4}

ordinal_owner_category = pd.CategoricalDtype(
```

```
    categories=list(owner_type_order.keys()), ordered=True
)
df.Owner_Type = df.Owner_Type.astype(ordinal_owner_category)
df.Owner_Type.cat.categories
```

```
[44]: Index(['First', 'Second', 'Third', 'Fourth & Above'], dtype='object')
```

9. Mileage

Fuel efficiency of the car (kmpl or km/kg)

```
[45]: df.Mileage.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Mileage
Non-Null Count Dtype
-----
5961 non-null object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[46]: df.Mileage.head()
```

```
[46]: ID
G4XLU0      17.0 kmpl
CRSHOS     21.43 kmpl
FUJ4X1      13.8 kmpl
QMVK6E      21.27 kmpl
4SWHFC      17.0 kmpl
Name: Mileage, dtype: object
```

```
[47]: df.Mileage = df.Mileage.replace("\\"N", None)
```

```
[48]: df[["_Mileage_Value", "_Mileage_Unit"]] = df.Mileage.str.split(expand=True)
df._Mileage_Unit.unique()
```

```
[48]: array(['kmpl', 'km/kg', None], dtype=object)
```

```
[49]: df._Mileage_Value = df._Mileage_Value.astype("float64")
df._Mileage_Unit = df._Mileage_Unit.astype("category")
```

kmpl and km/kg units are mixed.

kmpl and km/kg would have the same scale if the density was 1 (water). However, depending on the fuel type, the density can be different, thus the conversion factor too.

Luckily, Fuel_Type is available.

```
[50]: df.Fuel_Type.cat.categories
```

```
[50]: Index(['CNG', 'Diesel', 'Electric', 'LPG', 'Petrol'], dtype='object')

[51]: conversion_factors = {"CNG": 1.33, "Diesel": 1.20, "LPG": 1.85, "Petrol": 1.35}

df.Mileage = df._Mileage_Value.copy()

for fuel_type, factor in conversion_factors.items():
    mask = (df._Mileage_Unit == "km/kg") & (df.Fuel_Type == fuel_type)
    df.loc[mask, "Mileage"] = df.loc[mask, "_Mileage_Value"] * factor
df = df.drop(["_Mileage_Unit", "_Mileage_Value"], axis=1)

df.Mileage
```

```
[51]: ID
G4XLU0      17.00
CRSHOS      21.43
FUJ4X1      13.80
QMVK6E      21.27
4SWHFC      17.00
...
CWRWOT      13.93
Q7Z939      18.33
73KOPC      16.55
XEBBLO      12.05
LOLVST      18.60
Name: Mileage, Length: 5961, dtype: float64
```

10. Engine

Engine displacement in cubic centimeters (CC)

```
[52]: df.Engine.info()

<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Engine
Non-Null Count Dtype
-----
5961 non-null   object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[53]: df.Engine.head()
```

```
[53]: ID
G4XLU0      1405 CC
CRSHOS      1364 CC
FUJ4X1      1299 CC
```

```
QMVK6E      1396 CC
4SWHFC      1497 CC
Name: Engine, dtype: object
```

```
[54]: df.Engine = df.Engine.replace("\\\n", None)
```

```
[55]: df.Engine = pd.to_numeric(df.Engine.str.split(expand=True)[0])
df.Engine = df.Engine.astype("Int64")
df.Engine
```

```
[55]: ID
G4XLU0      1405
CRSHOS      1364
FUJ4X1      1299
QMVK6E      1396
4SWHFC      1497
...
CWRWOT      2179
Q7Z939      1968
73K0PC      1968
XEBBL0      2179
LOLVST      1197
Name: Engine, Length: 5961, dtype: Int64
```

11. Power

Maximum power output of the engine (bhp - brake horsepower)

```
[56]: df.Power.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Power
Non-Null Count   Dtype
-----
5961 non-null    object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[57]: df.Power = df.Power.replace("\\\n", None)
```

```
[58]: df.Power = pd.to_numeric(df.Power.str.split(expand=True)[0], errors="coerce")
df.Power
```

```
[58]: ID
G4XLU0      70.00
CRSHOS      87.20
FUJ4X1      70.00
```

```
QMVK6E      88.76
4SWHFC     118.00
...
CWRWOT     138.03
Q7Z939     167.70
73KOPC     147.51
XEBBLO     120.00
LOLVST      81.83
Name: Power, Length: 5961, dtype: float64
```

12. Colour

Exterior color of the vehicle

```
[59]: df.Colour.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Colour
Non-Null Count   Dtype
-----
5961 non-null    object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[60]: df.Colour.head()
```

```
[60]: ID
G4XLU0      Others
CRSHOS      Others
FUJ4X1      Others
QMVK6E      Black/Silver
4SWHFC      White
Name: Colour, dtype: object
```

```
[61]: df.Colour.unique()
```

```
[61]: array(['Others', 'Black/Silver', 'White', '\\N'], dtype=object)
```

```
[62]: df.Colour = df.Colour.replace("\\N", None)
```

```
[63]: df.Colour = df.Colour.astype("category")
df.Colour
```

```
[63]: ID
G4XLU0      Others
CRSHOS      Others
FUJ4X1      Others
```

```
QMVK6E    Black/Silver
4SWHFC      White
...
CWRWOT        Others
Q7Z939    Black/Silver
73KOPC    Black/Silver
XEBBLO        Others
LOLVST    Black/Silver
Name: Colour, Length: 5961, dtype: category
Categories (3, object): ['Black/Silver', 'Others', 'White']
```

13. Seats

Number of seating capacity

```
[64]: df.Seats.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: Seats
Non-Null Count Dtype
-----
5961 non-null   object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[65]: df.Seats.head()
```

```
[65]: ID
G4XLU0    5
CRSHOS    5
FUJ4X1    5
QMVK6E    5
4SWHFC    5
Name: Seats, dtype: object
```

```
[66]: df.Seats = df.Seats.replace("\\\n", None)
```

```
[67]: df.Seats = pd.to_numeric(df.Seats, errors="coerce")
df.Seats = df.Seats.astype("Int64")
df.Seats
```

```
[67]: ID
G4XLU0    5
CRSHOS    5
FUJ4X1    5
QMVK6E    5
4SWHFC    5
```

```
..  
CWRWOT    7  
Q7Z939     5  
73KOPC     5  
XEBBLO     8  
LOLVST      5  
Name: Seats, Length: 5961, dtype: Int64
```

14. No. of Doors

Number of doors in the vehicle

```
[68]: df["No. of Doors"].info()
```

```
<class 'pandas.core.series.Series'>  
Index: 5961 entries, G4XLU0 to LOLVST  
Series name: No. of Doors  
Non-Null Count Dtype  
-----  
5961 non-null    object  
dtypes: object(1)  
memory usage: 222.2+ KB
```

```
[69]: df.rename(columns={"No. of Doors": "Doors"}, inplace=True)
```

```
[70]: df.Doors
```

```
[70]: ID  
G4XLU0    4  
CRSHOS    4  
FUJ4X1    4  
QMVK6E    4  
4SWHFC    4  
..  
CWRWOT    5  
Q7Z939     4  
73KOPC     4  
XEBBLO     5  
LOLVST      4  
Name: Doors, Length: 5961, dtype: object
```

```
[71]: df.Doors.unique()
```

```
[71]: array(['4', '5', '\\N', '2', 4, 5, 2], dtype=object)
```

```
[72]: df.Doors = df.Doors.replace("\\N", None)
```

```
[73]: df.Doors = pd.to_numeric(df.Doors)
df.Doors = df.Doors.astype("Int64")
df.Doors
```

```
[73]: ID
G4XLU0    4
CRSHOS    4
FUJ4X1    4
QMVK6E    4
4SWHFC    4
...
CWRWOT    5
Q7Z939    4
73KOPC    4
XEBBL0    5
LOLVST    4
Name: Doors, Length: 5961, dtype: Int64
```

15. New_Price

Original price of the car when it was new (may contain missing values)

```
[74]: df.New_Price.info()
```

```
<class 'pandas.core.series.Series'>
Index: 5961 entries, G4XLU0 to LOLVST
Series name: New_Price
Non-Null Count   Dtype
-----
5961 non-null   object
dtypes: object(1)
memory usage: 222.2+ KB
```

```
[75]: df.New_Price = df.New_Price.replace("\N", None)
df[df.New_Price.isna()]
```

```
[75]:      Location  Year Kilometers_Driven Fuel_Type Transmission Owner_Type \
ID
G4XLU0  Coimbatore  2013        59138 Diesel     Manual    First
CRSHOS      Kochi  2013        81504 Diesel     Manual    First
FUJ4X1  Hyderabad  2007       92000 Petrol     Manual    First
QMVK6E      Kolkata 2012       33249 Diesel     Manual    First
4SWHFC  Bangalore  2011       65000 Petrol     Manual    First
...
...      ...   ...
CWRWOT  Bangalore  2011       80000 Diesel     Manual    First
Q7Z939      Kolkata 2011       42500 Diesel  Automatic  First
73KOPC  Bangalore  2014       37600 Diesel  Automatic Second
XEBBL0  Bangalore  2011       73000 Diesel     Manual    First
```

LOLVST	Coimbatore	2017		14618	Petrol	Manual	First		
ID	Mileage	Engine	Power	Colour	Seats	Doors	New_Price	Price	\
G4XLU0	17.00	1405	70.00	Others	5	4	None	2.58	
CRSHOS	21.43	1364	87.20	Others	5	4	None	6.53	
FUJ4X1	13.80	1299	70.00	Others	5	4	None	1.25	
QMVK6E	21.27	1396	88.76	Black/Silver	5	4	None	3.25	
4SWHFC	17.00	1497	118.00	White	5	4	None	5.20	
...	
CWRWOT	13.93	2179	138.03	Others	7	5	None	Nan	
Q7Z939	18.33	1968	167.70	Black/Silver	5	4	None	Nan	
73KOPC	16.55	1968	147.51	Black/Silver	5	4	None	Nan	
XEBBLO	12.05	2179	120.00	Others	8	5	None	Nan	
LOLVST	18.60	1197	81.83	Black/Silver	5	4	None	Nan	

ID	Brand	Model	Age
G4XLU0	Tata	Indigo	7
CRSHOS	Toyota	Corolla	7
FUJ4X1	Ford	Ikon	13
QMVK6E	Hyundai	I20	8
4SWHFC	Honda	City	9
...
CWRWOT	Tata	Safari	9
Q7Z939	Volkswagen	Passat	9
73KOPC	Audi	A4	6
XEBBLO	Mahindra	Scorpio	9
LOLVST	Hyundai	I20	3

[5137 rows x 17 columns]

5137 out of 5961 samples are missing New_Price.

[76]: df.New_Price[df.New_Price.notna()]

[76]: ID

BOUPCL	79.43 Lakh
2FFBRO	21.72 Lakh
9OEINM	8.17 Lakh
QU3AAV	95.13 Lakh
4QPA01	33.36 Lakh
	...
74N7UN	8.82 Lakh
RKGVC	8.27 Lakh
SX4HME	13.72 Lakh
V7CA4M	15.94 Lakh
GDLPH2	7.85 Lakh

```
Name: New_Price, Length: 824, dtype: object
```

```
[77]: df[["_New_Price_Value", "_New_Price_Unit"]] = df.New_Price.str.  
      ↪split(expand=True)  
df._New_Price_Value = pd.to_numeric(df._New_Price_Value)  
df._New_Price_Unit = df._New_Price_Unit.astype("category")
```

```
[78]: df._New_Price_Unit.cat.categories
```

```
[78]: Index(['Cr', 'Lakh'], dtype='object')
```

Lakh and Cr unit is mixed. Standardizing to Lakh. 1 Cr is 100 lakh.

```
[79]: cr_mask = df._New_Price_Unit == "Cr"  
df.loc[cr_mask, "_New_Price_Value"] = df.loc[cr_mask, "_New_Price_Value"] * 100  
df.loc[cr_mask, "_New_Price_Unit"] = "Lakh"
```

```
[80]: df.New_Price = df._New_Price_Value.copy()  
df.New_Price[df.New_Price.notna()]
```

```
[80]: ID  
B0UPCL    79.43  
2FFBRO    21.72  
9OEINM     8.17  
QU3AAV    95.13  
4QPA01    33.36  
...  
74N7UN     8.82  
RKGVCVP   8.27  
SX4HME    13.72  
V7CA4M    15.94  
GDLPH2    7.85
```

```
Name: New_Price, Length: 824, dtype: float64
```

```
[81]: df.drop(columns=["_New_Price_Value", "_New_Price_Unit"], inplace=True)
```

16 Price

Target variable: Current selling price of the used car (in lakhs - 1 lakh = 100,000 Indian Rupees)

```
[82]: df.Price.info()
```

```
<class 'pandas.core.series.Series'>  
Index: 5961 entries, G4XLU0 to LOLVST  
Series name: Price  
Non-Null Count Dtype  
-----  
4470 non-null   float64
```

```
dtypes: float64(1)
memory usage: 222.2+ KB
```

```
[83]: df.Price
```

```
[83]: ID
G4XLU0    2.58
CRSHOS    6.53
FUJ4X1    1.25
QMVK6E    3.25
4SWHFC    5.20
...
CWRWOT    NaN
Q7Z939    NaN
73KOPC    NaN
XEBBL0    NaN
LOLVST    NaN
Name: Price, Length: 5961, dtype: float64
```

0.1.1 Impute

```
[84]: df_train = df.loc[df_train.ID, :]
df_test = df.loc[df_test.ID, :]
```

```
[85]: print("# of missing values in df_train for each features (in %)")
for feature in df_train.columns:
    missing_ratio = len(df_train[df_train[feature].isna()]) / len(df_train) * 100
    print(
        f'{feature.ljust(max(len(f) + 1 for f in df_train.columns))}: {{:.2f}}'.
        format(missing_ratio).rjust(6)}% {'<-' if missing_ratio > 1.0 else ''}")

print("\n# of missing values in df_test for each features (in %)")
for feature in df_test.columns:
    missing_ratio = len(df_test[df_test[feature].isna()]) / len(df_test) * 100
    print(
        f'{feature.ljust(max(len(f) + 1 for f in df_test.columns))}: {{:.2f}}'.
        format(missing_ratio).rjust(6)}% {'<-' if missing_ratio > 1.0 else ''}'')

# of missing values in df_train for each features (in %)
Location      : 0.18%
Year          : 0.04%
Kilometers_Driven : 0.11%
Fuel_Type     : 0.00%
```

```

Transmission      : 0.40%
Owner_Type        : 0.22%
Mileage           : 0.04%
Engine             : 0.25%
Power              : 2.37% <-
Colour             : 0.18%
Seats               : 0.04%
Doors               : 0.02%
New_Price          : 86.31% <-
Price               : 0.00%
Brand               : 0.00%
Model               : 0.00%
Age                 : 0.04%

# of missing values in df_test for each features (in %)
Location          : 0.20%
Year                : 0.00%
Kilometers_Driven : 0.20%
Fuel_Type          : 0.00%
Transmission       : 0.60%
Owner_Type          : 0.34%
Mileage             : 0.00%
Engine              : 0.40%
Power              : 1.95% <-
Colour             : 0.20%
Seats               : 0.20%
Doors               : 0.00%
New_Price          : 85.78% <-
Price               : 100.00% <-
Brand               : 0.00%
Model               : 0.00%
Age                 : 0.00%

```

Features except Power and New_Price has less than 1% missing values, and thus rows with missing features other than Power or New_Price can be safely dropped.

```
[86]: print(f"# of training data samples before dropping: {len(df_train)}")
df_train.dropna(
    subset=list(set(df_train.columns) - {"Power", "New_Price"}), inplace=True
)
print(f"# of training data samples after dropping: {len(df_train)}")

df_train
```

```
# of training data samples before dropping: 4470
# of training data samples after dropping: 4428
```

[86] :

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	\		
ID									
G4XLU0	Coimbatore	2013	59138	Diesel	Manual	First			
CRSHOS	Kochi	2013	81504	Diesel	Manual	First			
FUJ4X1	Hyderabad	2007	92000	Petrol	Manual	First			
QMVK6E	Kolkata	2012	33249	Diesel	Manual	First			
4SWHFC	Bangalore	2011	65000	Petrol	Manual	First			
...			
TR7SLB	Kochi	2016	51884	Diesel	Manual	First			
QB41QE	Kolkata	2016	27210	Diesel	Manual	First			
ODG8N7	Pune	2015	52000	Diesel	Automatic	First			
EV2ZBX	Delhi	2013	56000	Petrol	Manual	First			
J2RCU8	Bangalore	2014	52000	Diesel	Automatic	First			
	Mileage	Engine	Power	Colour	Seats	Doors	New_Price	Price	\
ID									
G4XLU0	17.00	1405	70.00	Others	5	4	NaN	2.58	
CRSHOS	21.43	1364	87.20	Others	5	4	NaN	6.53	
FUJ4X1	13.80	1299	70.00	Others	5	4	NaN	1.25	
QMVK6E	21.27	1396	88.76	Black/Silver	5	4	NaN	3.25	
4SWHFC	17.00	1497	118.00	White	5	4	NaN	5.20	
...	
TR7SLB	16.00	2179	140.00	White	7	5	NaN	12.46	
QB41QE	27.30	1498	98.60	Others	5	4	NaN	5.85	
ODG8N7	12.70	2179	187.70	White	5	4	NaN	39.75	
EV2ZBX	24.70	796	47.30	Others	5	4	NaN	2.10	
J2RCU8	12.00	2987	224.00	Black/Silver	7	5	NaN	49.00	
	Brand	Model	Age						
ID									
G4XLU0	Tata	Indigo	7						
CRSHOS	Toyota	Corolla	7						
FUJ4X1	Ford	Ikon	13						
QMVK6E	Hyundai	I20	8						
4SWHFC	Honda	City	9						
...						
TR7SLB	Mahindra	Xuv500	4						
QB41QE	Honda	Jazz	4						
ODG8N7	Land_Rover	Range	5						
EV2ZBX	Maruti	Alto	7						
J2RCU8	Mercedes_Benz	Gl-Class	6						

[4428 rows x 17 columns]

[87] : df_test

[87] :		Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	\	
ID									
INQOD6	Pune	2012		63400	Diesel	Manual	First		
S7ZJIY	Chennai	2008		89000	Diesel	Automatic	Second		
CZ59WU	NAN	2010		<NA>	Diesel	Automatic	First		
P6II8S	Mumbai	2017		32000	Petrol	Automatic	First		
500X2V	Coimbatore	2012		77283	Petrol	Manual	First		
...		
CWRWOT	Bangalore	2011		80000	Diesel	Manual	First		
Q7Z939	Kolkata	2011		42500	Diesel	Automatic	First		
73KOPC	Bangalore	2014		37600	Diesel	Automatic	Second		
XEBBL0	Bangalore	2011		73000	Diesel	Manual	First		
LOLVST	Coimbatore	2017		14618	Petrol	Manual	First		
	Mileage	Engine	Power	Colour	Seats	Doors	New_Price	Price	\
ID									
INQOD6	17.80	1399	67.00	Black/Silver	5	4	NaN	NaN	
S7ZJIY	16.07	1995	181.00	Black/Silver	4	4	NaN	NaN	
CZ59WU	12.40	2698	179.50	Others	5	4	NaN	NaN	
P6II8S	14.84	1598	103.52	Black/Silver	5	4	13.7	NaN	
500X2V	17.00	1497	118.00	Black/Silver	5	4	NaN	NaN	
...	
CWRWOT	13.93	2179	138.03	Others	7	5	NaN	NaN	
Q7Z939	18.33	1968	167.70	Black/Silver	5	4	NaN	NaN	
73KOPC	16.55	1968	147.51	Black/Silver	5	4	NaN	NaN	
XEBBL0	12.05	2179	120.00	Others	8	5	NaN	NaN	
LOLVST	18.60	1197	81.83	Black/Silver	5	4	NaN	NaN	
	Brand	Model	Age						
ID									
INQOD6	Ford	Fiesta	8						
S7ZJIY	BMW		3	12					
CZ59WU	Audi		A6	10					
P6II8S	Skoda	Rapid		3					
500X2V	Honda	City		8					
...					
CWRWOT	Tata	Safari		9					
Q7Z939	Volkswagen	Passat		9					
73KOPC	Audi		A4	6					
XEBBL0	Mahindra	Scorpio		9					
LOLVST	Hyundai	I20		3					

[1491 rows x 17 columns]

A.3 Feature Engineering Notebook (*feature-engineering.ipynb*)

feature-engineering

November 17, 2025

0.0.1 Feature Engineering

```
[1]: %load_ext autoreload  
%autoreload 2
```

```
[2]: from mldl_hw3.preprocessing import DataLoader  
from mldl_hw3.consts import Brand  
from mldl_hw3.experiment import Experiment, ExperimentConfig  
  
from typing import Optional  
  
from xgboost import XGBRegressor  
from sklearn.pipeline import Pipeline  
from sklearn.base import BaseEstimator, TransformerMixin  
from sklearn.compose import TransformedTargetRegressor  
from sklearn.preprocessing import OneHotEncoder, StandardScaler  
from sklearn.linear_model import LinearRegression  
from sklearn.cluster import KMeans  
from category_encoders import TargetEncoder  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np
```

```
[3]: df_train, df_test = DataLoader("../dataset").load()
```

```
[4]: df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 4428 entries, G4XLU0 to J2RCU8  
Data columns (total 15 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --     
 0   Name            4428 non-null    category  
 1   Location        4428 non-null    category  
 2   Year            4428 non-null    Int64  
 3   Kilometers_Driven 4428 non-null    Int64  
 4   Fuel_Type        4428 non-null    category  
 5   Transmission     4428 non-null    category
```

```

6   Owner_Type          4428 non-null  category
7   Mileage              4428 non-null  float64
8   Engine                4428 non-null  Int64
9   Power                 4336 non-null  float64
10  Colour               4428 non-null  category
11  Seats                  4428 non-null  Int64
12  Doors                  4428 non-null  Int64
13  New_Price             601 non-null  float64
14  Price                 4428 non-null  float64
dtypes: Int64(5), category(6), float64(4)
memory usage: 408.6+ KB

```

```
[5]: X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.drop(columns=["Price"])
```

0.0.2 Baseline

```
[6]: class CategoricalEncoder(TransformerMixin, BaseEstimator):
    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "CategoricalEncoder":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        for feature in X.select_dtypes(["category"]):
            X[feature] = X[feature].cat.codes

        return X
```

```
[7]: def build_baseline_pipeline(**model_params) -> Pipeline:
    return Pipeline(
        [
            ("encoder", CategoricalEncoder()),
            ("model", XGBRegressor(**model_params)),
        ]
    )
```

```
[8]: baseline_exp = Experiment(
    ExperimentConfig(name="baseline", pipeline=build_baseline_pipeline())
)

baseline_exp_result = baseline_exp.run(X_train, y_train, X_test)
```

```
[Experiment: baseline]
Cross-validating (5-folds)...
```

```

CV score: 0.1606 ± 0.0163
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/baseline.csv
Experiment complete

```

Brand extraction from Name Form the analysis on `preprocessing.ipynb`, we found that `Name` feature has too many categories. The car's name usually consists of brand and model. From common knowledge, we know that the car's brand has significant impact on forming its price. Based on this domain knowledge and to reduce category, we extract Brand from Name.

```

[9]: class BrandModelExtractor(TransformerMixin, BaseEstimator):
    def __init__(self,
                 name_col: str = "Name",
                 brand_col: str = "Brand",
                 model_col: str = "Model",
                 brand_enum=Brand,
                 ):
        self.name_col = name_col
        self.brand_col = brand_col
        self.model_col = model_col
        self.brand_enum = brand_enum

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> Union["BrandModelExtractor", None]:
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        names = X[self.name_col].str.title()
        brands = pd.Series([None] * len(names), index=names.index, dtype=object)
        models = pd.Series([None] * len(names), index=names.index, dtype=object)

        for brand in Brand:
            condition = names.str.startswith(brand.value, na=False) & brands.isna()

            if condition.any():
                brands.loc[condition] = brand.name
                matched_names = names.loc[condition]
                residuals = matched_names.str[len(brand.value):].str.strip()
                models.loc[condition] = residuals.where(residuals != "", None)

        brands = brands.rename("Brand").astype("category")

```

```

models = models.rename("Model").astype("category")

X[self.brand_col] = brands
X[self.model_col] = models

return X

```

[10]:

```

X_train = df_train.copy()
y_train = X_train.pop("Price")

df_train_bm = pd.DataFrame(BrandModelExtractor().fit_transform(X_train, y_train))
df_train_bm

```

[10]:

ID	Name	Location	Year	Kilometers_Driven	Fuel_Type
G4XLU0	Tata Indigo	Coimbatore	2013	59138	Diesel
CRSHOS	Toyota Corolla	Kochi	2013	81504	Diesel
FUJ4X1	Ford Ikon	Hyderabad	2007	92000	Petrol
QMVK6E	Hyundai i20	Kolkata	2012	33249	Diesel
4SWHFC	Honda City	Bangalore	2011	65000	Petrol
...
TR7SLB	Mahindra XUV500	Kochi	2016	51884	Diesel
QB41QE	Honda Jazz	Kolkata	2016	27210	Diesel
ODG8N7	Land Rover Range	Pune	2015	52000	Diesel
EV2ZBX	Maruti Alto	Delhi	2013	56000	Petrol
J2RCU8	Mercedes-Benz GL-Class	Bangalore	2014	52000	Diesel

ID	Transmission	Owner_Type	Mileage	Engine	Power	Colour	Seats
G4XLU0	Manual	First	17.00	1405	70.00	Others	5
CRSHOS	Manual	First	21.43	1364	87.20	Others	5
FUJ4X1	Manual	First	13.80	1299	70.00	Others	5
QMVK6E	Manual	First	21.27	1396	88.76	Black/Silver	5
4SWHFC	Manual	First	17.00	1497	118.00	White	5
...
TR7SLB	Manual	First	16.00	2179	140.00	White	7
QB41QE	Manual	First	27.30	1498	98.60	Others	5
ODG8N7	Automatic	First	12.70	2179	187.70	White	5
EV2ZBX	Manual	First	24.70	796	47.30	Others	5
J2RCU8	Automatic	First	12.00	2987	224.00	Black/Silver	7

ID	Doors	New_Price	Brand	Model
G4XLU0	4	NaN	Tata	Indigo
CRSHOS	4	NaN	Toyota	Corolla
FUJ4X1	4	NaN	Ford	Ikon

QMVK6E	4	NaN	Hyundai	I20
4SWHFC	4	NaN	Honda	City
...
TR7SLB	5	NaN	Mahindra	Xuv500
QB41QE	4	NaN	Honda	Jazz
ODG8N7	4	NaN	Land_Rover	Range
EV2ZBX	4	NaN	Maruti	Alto
J2RCU8	5	NaN	Mercedes_Benz	G1-Class

[4428 rows x 16 columns]

Note that Brand and Model has an hierarchical relationship.

```
[11]: display(
    df_train_bm[["Brand", "Model"]].groupby(["Brand", "Model"], observed=True).
    ↪count()
)
```

Empty DataFrame
Columns: []
Index: [(Audi, A3), (Audi, A4), (Audi, A6), (Audi, A7), (Audi, A8), (Audi, Q3),
(Audi, Q5), (Audi, Q7), (Audi, Rs5), (Audi, Tt), (BMW, 1), (BMW, 3), (BMW, 5),
(BMW, 6), (BMW, 7), (BMW, X1), (BMW, X3), (BMW, X5), (BMW, X6), (BMW, Z4),
(Bentley, Continental), (Chevrolet, Aveo), (Chevrolet, Beat), (Chevrolet,
Captiva), (Chevrolet, Cruze), (Chevrolet, Enjoy), (Chevrolet, Optra),
(Chevrolet, Sail), (Chevrolet, Spark), (Datsun, Go), (Datsun, Redi), (Datsun,
Redi-Go), (Fiat, Avventura), (Fiat, Grande), (Fiat, Linea), (Fiat, Petra),
(Fiat, Punto), (Fiat, Siena), (Force, One), (Ford, Aspire), (Ford, Ecosport),
(Ford, Endeavour), (Ford, Fiesta), (Ford, Figo), (Ford, Freestyle), (Ford,
Fusion), (Ford, Ikon), (Honda, Accord), (Honda, Amaze), (Honda, Br-V), (Honda,
Brio), (Honda, Brv), (Honda, City), (Honda, Civic), (Honda, Cr-V), (Honda,
Jazz), (Honda, Mobilio), (Honda, Wr-V), (Honda, Wrv), (Hyundai, Accent),
(Hyundai, Creta), (Hyundai, Elantra), (Hyundai, Elite), (Hyundai, Eon),
(Hyundai, Getz), (Hyundai, Grand), (Hyundai, I10), (Hyundai, I20), (Hyundai,
Santa), (Hyundai, Santro), (Hyundai, Sonata), (Hyundai, Tucson), (Hyundai,
Verna), (Hyundai, Xcent), (Isuzu, D-Max), (Isuzu, Mux), (Jaguar, F), (Jaguar,
Xe), (Jaguar, Xf), (Jaguar, Xj), (Jeep, Compass), (Lamborghini, Gallardo),
(Land_Rover, Discovery), (Land_Rover, Freelander), (Land_Rover, Range),
(Mahindra, Bolero), (Mahindra, Jeep), (Mahindra, Kuv), (Mahindra, Logan),
(Mahindra, Nuvosport), (Mahindra, Quanto), (Mahindra, Renault), (Mahindra,
Scorpio), (Mahindra, Ssangyong), (Mahindra, Thar), (Mahindra, Tuv), (Mahindra,
Verito), (Mahindra, Xuv300), (Mahindra, Xuv500), (Mahindra, Xylo), ...]

[204 rows x 0 columns]

```
[12]: df_train_bm.Brand.unique()
```

```
[12]: ['Tata', 'Toyota', 'Ford', 'Hyundai', 'Honda', ..., 'Smart', 'Isuzu', 'Volvo',
'Bentley', 'Lamborghini']
Length: 29
Categories (29, object): ['Audi', 'BMW', 'Bentley', 'Chevrolet', ..., 'Tata',
'Toyota', 'Volkswagen', 'Volvo']
```

```
[13]: df_train_bm.Brand.value_counts()
```

```
[13]: Brand
Maruti           884
Hyundai          817
Honda            462
Toyota           309
Mercedes_Benz    239
Volkswagen       225
Ford              217
Mahindra          199
BMW               191
Audi              177
Tata              140
Skoda             117
Renault            117
Chevrolet          82
Nissan             64
Land_Rover         43
Jaguar             28
Fiat               23
Mini               21
Mitsubishi         20
Porsche             14
Volvo              14
Jeep                11
Datsun              7
Isuzu               3
Lamborghini          1
Force                1
Smart                1
Bentley               1
Name: count, dtype: int64
```

```
[14]: df_train_bm.Model.unique()
```

```
[14]: ['Indigo', 'Corolla', 'Ikon', 'I20', 'City', ..., 'Cls-Class', 'Gallardo',
'Wr-V', 'Outlander', 'Xc90']
Length: 201
Categories (201, object): ['1', '3', '5', '6', ..., 'Yeti', 'Z4', 'Zen', 'Zest']
```

```
[15]: with pd.option_context("display.max_rows", None):
    display(df_train_bm.Model.value_counts())
```

Model	
Swift	253
City	206
I20	183
Verna	129
Grand	125
Innova	121
I10	120
Wagon	113
Alto	107
Polo	107
Xuv500	89
New	80
Amaze	80
Fortuner	77
Vento	76
3	75
Ecosport	72
Figo	71
Creta	69
Duster	65
E-Class	63
Ertiga	58
A4	55
Santro	53
Ciaz	51
Corolla	51
Etios	50
Ritz	48
5	47
Baleno	45
Brio	44
Jazz	44
Eon	44
Celerio	44
Scorpio	40
Xcent	39
A6	36
Vitara	35
Kwid	34
Superb	34
Indigo	29
Beat	29
Civic	27
Q7	27

Endeavour	27
Indica	27
Rapid	26
Q5	25
Fiesta	25
X1	24
Zen	24
Octavia	23
Micra	23
Range	23
Q3	22
Dzire	21
Accord	21
Laura	20
Sx4	20
Xf	20
Cr-V	19
Cooper	19
Sunny	19
Zest	19
M-Class	18
Nano	18
Terrano	18
Gla	17
Jetta	17
Ameo	17
X5	16
S	15
Xylo	15
Bolero	15
Ikon	15
Eeco	15
Pajero	15
Aveo	13
Elantra	13
Santa	13
A-Star	13
Cruze	13
Manza	12
Linea	12
Accent	12
Compass	11
Discovery	10
Mobilio	10
Freelander	10
Gl-Class	10
Gle	9
Fabia	9

Cla	9
X3	9
Spark	8
B	8
Optra	8
Tiago	8
Omni	8
Kuv	8
800	7
Cayenne	7
Ssangyong	7
Elite	7
Tuv	6
Getz	6
Camry	6
Glc	6
Safari	6
Sail	6
Pulse	5
Xj	5
6	5
Yeti	5
X6	5
Esteem	5
7	5
Passat	5
Grande	5
Scala	4
Sonata	4
S60	4
Brv	4
S-Class	4
Quanto	4
Panamera	4
Aspire	4
Ignis	4
Xc60	4
A	4
V40	4
Avventura	3
Thar	3
Tigor	3
Tucson	3
Bolt	3
Verito	3
A7	3
A3	3
Wrv	3

S-Cross	3
1	3
Redi-Go	3
Enjoy	3
Br-V	3
Fluence	3
R-Class	3
Captur	3
Qualis	3
Koleos	3
Go	3
Hexa	3
Renault	2
X-Trail	2
Logan	2
Lancer	2
A8	2
Jeep	2
Gls	2
Crosspolo	2
D-Max	2
Cayman	2
Xe	2
Xenon	2
Freestyle	2
Z4	2
Rs5	2
Tt	2
Captiva	2
Nuvosport	2
C-Class	2
Sumo	2
Fusion	1
Xc90	1
Beetle	1
Punto	1
Gallardo	1
Xuv300	1
Slc	1
F	1
Fortwo	1
S80	1
Boxster	1
Redi	1
Siena	1
Nexon	1
Slk-Class	1
Mux	1

```

Countryman      1
Platinum       1
Continental    1
Wr-V           1
Petra          1
Outlander      1
Clubman         1
One             1
Cls-Class      1
Cedia           1
Teana           1
Montero         1
Evalia          1
Name: count, dtype: int64

```

Model feature has too high cardinality, and there are non-negligible amount of categories with very small sample. Since we already have Brand, we could just drop Model. Alternatively, we could try target encoding.

0. Baseline (only Name)
1. Categorical Brand and Model added

```
[16]: extract_brand_model_exp = Experiment(
    ExperimentConfig(
        name="extract-brand-model",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

extract_brand_model_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```

[Experiment: extract-brand-model]
Cross-validating (5-folds)...
CV score: 0.1905 ± 0.0190
    +0.0299  +0.0027 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/extract-brand-model.csv
Experiment complete

```

2. Drop Model

```
[17]: class FeatureDropper(TransformerMixin, BaseEstimator):
    def __init__(self, cols: list[str]):
        self.cols = cols

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "FeatureDropper":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        return X.drop(columns=self.cols)
```

```
[18]: drop_model_exp = Experiment(
    ExperimentConfig(
        name="drop-model",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                ("drop_model", FeatureDropper(["Model"])),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

drop_model_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

[Experiment: drop-model]
Cross-validating (5-folds)...
CV score: 0.1606 ± 0.0163
+0.0000 +0.0000 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/drop-model.csv
Experiment complete

3. Drop Name

```
[19]: drop_name_exp = Experiment(
    ExperimentConfig(
        name="drop-name",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                ("drop_name", FeatureDropper(["Name"])),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        )
)
```

```

        ),
    )
)

drop_name_exp.run(X_train, y_train, X_test, baseline_exp_result);

```

[Experiment: drop-name]
Cross-validating (5-folds)...
CV score: 0.2379 ± 0.0157
+0.0772 -0.0006 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/drop-name.csv
Experiment complete

4. Target encoding Model. (Leave Brand categorical)

```
[20]: model_te_exp = Experiment(
    ExperimentConfig(
        name="model-te",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                ("target_encode", TargetEncoder(cols=["Model"])),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

model_te_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

[Experiment: model-te]
Cross-validating (5-folds)...
CV score: 0.1523 ± 0.0107
-0.0083 -0.0056 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/model-te.csv
Experiment complete

5. Target encoding Brand and Model.

```
[21]: brand_model_te_exp = Experiment(
    ExperimentConfig(
        name="brand-model-te",
```

```

pipeline=Pipeline(
    [
        ("extract_brand_model", BrandModelExtractor()),
        ("target_encode", TargetEncoder(cols=["Brand", "Model"])),
        ("category_encode", CategoricalEncoder()),
        ("model", XGBRegressor()),
    ],
),
)
)

brand_model_te_exp.run(X_train, y_train, X_test, baseline_exp_result);

```

[Experiment: brand-model-te]
Cross-validating (5-folds)...
CV score: 0.1496 ± 0.0170
-0.0110 +0.0007 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/brand-model-te.csv
Experiment complete

6. Target encoding Brand, Model, and Name.

```

[22]: brand_model_name_te_exp = Experiment(
    ExperimentConfig(
        name="brand-model-name-te",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name"]),
                ),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ],
        ),
    )
)

brand_model_name_te_exp.run(X_train, y_train, X_test, baseline_exp_result);

```

[Experiment: brand-model-name-te]
Cross-validating (5-folds)...
CV score: 0.1505 ± 0.0136
-0.0101 -0.0027 compared to baseline (Negative is better)
Training on full training set...

```

Creating submission on test set...
Submission created: artifacts/experiment-results/brand-model-name-te.csv
Experiment complete

```

Target encoding Brand, Model, and Name yields the biggest gain.

Thinking deeper on the hierarchical relationship of Brand and Model, the difference of Model target encoded value and Brand target encoded value can tell how high class is the car in that brand. Test whether this helps.

```
[23]: class ModelPremiumEncoder(TransformerMixin, BaseEstimator):
    def __init__(self,
                 brand_col: str = "Brand",
                 model_col: str = "Model",
                 premium_col: str = "Model_Premium",
                 ):
        self.brand_col = brand_col
        self.model_col = model_col
        self.premium_col = premium_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series] = None) -> "ModelPremiumEncoder":
        return self

    def transform(self, X: pd.DataFrame):
        X = X.copy()
        X[self.premium_col] = X[self.model_col] - X[self.brand_col]
        return X
```

```
[24]: X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.copy().drop(columns=["Price"])

brand_model_name_mp_te_exp = Experiment(
    ExperimentConfig(
        name="model-premium",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name"]),
                ),
                ("model_premium", ModelPremiumEncoder()),
                ("category_encode", CategoricalEncoder()),
            ]
        )
    )
)
```

```

        ("model", XGBRegressor()),
    ],
),
)

brand_model_name_mp_te_exp.run(X_train, y_train, X_test, baseline_exp_result);

```

```

[Experiment: model-premium]
Cross-validating (5-folds)...
CV score: 0.1485 ± 0.0161
    -0.0121 -0.0002 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/model-premium.csv
Experiment complete

```

Model_Premium improved the performance slightly. However, the gain is marginal and standard deviation is worse than without it. I think this is not worth it and will stick with target encoded Brand, Model, and Name.

Location

```
[25]: df_train.Location.value_counts()
```

```

[25]: Location
Mumbai      586
Hyderabad   546
Coimbatore  478
Kochi        475
Pune         465
Delhi        420
Kolkata     384
Chennai      362
Jaipur       315
Bangalore    237
Ahmedabad   160
Name: count, dtype: int64

```

There is no location with significantly small datapoints.

Since location can influence car price due to regional demand, wealth, urban/rural, taxes, ... target encoding could provide a valuable signal of this regional effect.

1. Categorical Location (Same as baseline)

```
[26]: baseline_exp.run(X_train, y_train);
```

```

[Experiment: baseline]
Cross-validating (5-folds)...

```

```
CV score: 0.1606 ± 0.0163
Training on full training set...
Experiment complete
```

1. Target encoded Location

```
[27]: X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.copy().drop(columns=["Price"])

location_te_exp = Experiment(
    ExperimentConfig(
        name="location target encoding",
        pipeline=Pipeline(
            [
                (
                    "target_encode",
                    TargetEncoder(cols=["Location"]),
                ),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

location_te_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```
[Experiment: location target encoding]
Cross-validating (5-folds)...
CV score: 0.1559 ± 0.0162
    -0.0047 -0.0001 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/location target encoding.csv
Experiment complete
```

It does help to target encode Location.

Combining with the Name feature engineering.

```
[28]: name_location_best_exp = Experiment(
    ExperimentConfig(
        name="name, location best feature engineering",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
```

```

        (
            "target_encode",
            TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
        ),
        ("category_encode", CategoricalEncoder()),
        ("model", XGBRegressor()),
    ]
),
)
)

name_location_best_exp.run(X_train, y_train, X_test, baseline_exp_result);

[Experiment: name, location best feature engineering]
Cross-validating (5-folds)...
CV score: 0.1480 ± 0.0155
-0.0127 -0.0008 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/name, location best feature
engineering.csv
Experiment complete

```

Combined with engineered Name, it further increased the performance.

- brand-model-name-te: 0.1505 ± 0.0136
- brand-model-name-location-te: 0.1480 ± 0.0155

Year

```
[29]: df_train.Year.value_counts()
```

```
[29]: Year
2014    571
2016    564
2015    558
2013    484
2017    443
2012    411
2011    342
2010    247
2018    216
2009    146
2008    123
2007     93
2019     75
2006     55
2005     44
```

```

2004    21
2002    12
2003    11
2001     5
2000     3
1999     2
1998     2
Name: count, dtype: Int64

```

As explored in `preprocessing.ipynb`, calculating the `Age` might be more relevant in terms of price. Assuming that current year is 2020, I should try transforming `Year` to `Age`.

```
[30]: class YearToAgeTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, year_col: str = "Year", age_col: str = "Age", current_year: int = 2020):
        self.year_col = year_col
        self.age_col = age_col
        self.current_year = current_year

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series] = None) -> "YearToAgeTransformer":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.age_col] = self.current_year - X[self.year_col]
        X.drop(columns=[self.year_col], inplace=True)
        return X
```

0. `Year` (Same as baseline)

1. `Year` transformed to `Age`

```
[31]: year_to_age_transform_exp = Experiment(
    ExperimentConfig(
        name="year-to-age-transform",
        pipeline=Pipeline(
            [
                ("transform", YearToAgeTransformer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)
```

```

year_to_age_transform_exp.run(X_train, y_train, X_test, baseline_exp_result);

[Experiment: year-to-age-transform]
Cross-validating (5-folds)...
CV score: 0.1602 ± 0.0154
    -0.0004 -0.0009 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/year-to-age-transform.csv
Experiment complete

```

This transformation does not necessarily add more information but makes it more interpretable and nice to work with other features.

```

[32]: name_location_year_exp = Experiment(
    ExperimentConfig(
        name="name-location-year-engineered",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

name_location_year_exp.run(X_train, y_train, X_test, baseline_exp_result);

[Experiment: name-location-year-engineered]
Cross-validating (5-folds)...
CV score: 0.1479 ± 0.0159
    -0.0127 -0.0004 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/name-location-year-
engineered.csv
Experiment complete

```

Together with the Name and Location engineered, transforming Year to Age did not help at all. It might be due to Name -> Brand, Model, Name being the dominant feature. However, I will keep this

transformation since it makes the feature more interpretable while not hurting the performance. It could make this feature nicer to work with when analyzing interactions.

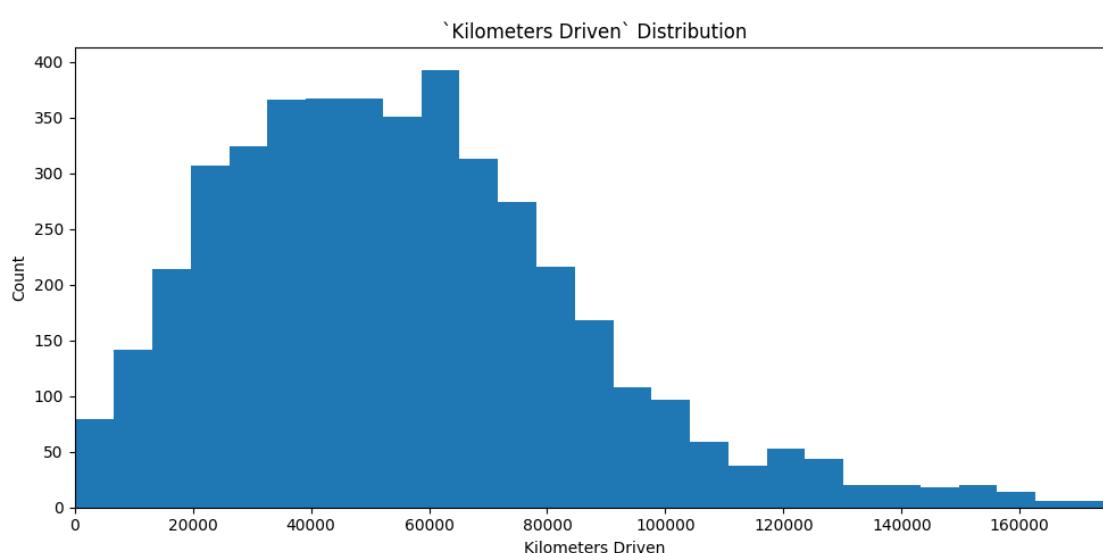
Kilometers Driven

```
[33]: df_train.Kilometers_Driven
```

```
[33]: ID
G4XLU0    59138
CRSHOS    81504
FUJ4X1    92000
QMVK6E    33249
4SWHFC    65000
...
TR7SLB    51884
QB41QE    27210
ODG8N7    52000
EV2ZBX    56000
J2RCU8    52000
Name: Kilometers_Driven, Length: 4428, dtype: Int64
```

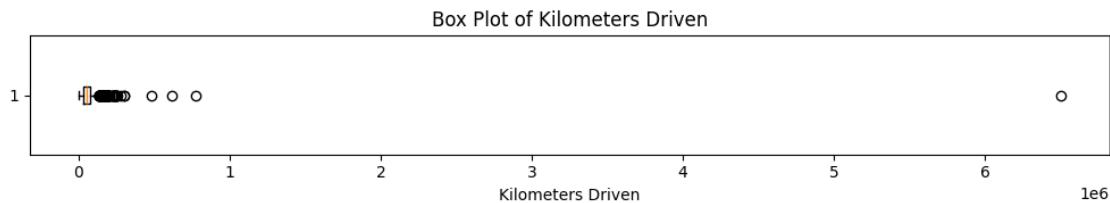
```
[34]: plt.figure(figsize=(10, 5))
plt.hist(df_train.Kilometers_Driven, bins=1000)
limit = np.percentile(df_train.Kilometers_Driven, 99)
plt.xlim(0, limit)
plt.xlabel("Kilometers Driven")
plt.ylabel("Count")
plt.title("`Kilometers Driven` Distribution")

plt.tight_layout()
plt.show()
```



This looks like a Gamma distribution. It makes sense because if we model the `Kilometers_Driven` as an accumulation of daily driving (with random noise added to consistent daily driving), it will yield a Gamma distribution.

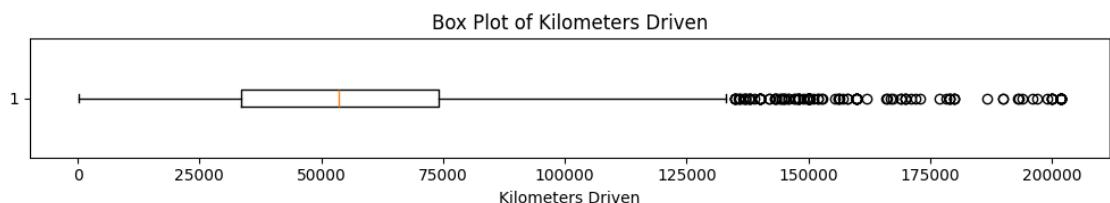
```
[35]: plt.figure(figsize=(10, 2))
plt.boxplot(df_train["Kilometers_Driven"], vert=False)
plt.xlabel("Kilometers Driven")
plt.title("Box Plot of Kilometers Driven")
plt.tight_layout()
plt.show()
```



There are clearly some extreme impossible data points. They should be clipped.

```
[36]: df_train.Kilometers_Driven = df_train.Kilometers_Driven.clip(
    upper=int(df_train.Kilometers_Driven.quantile(0.995))
)
df_test.Kilometers_Driven = df_test.Kilometers_Driven.clip(
    upper=int(df_test.Kilometers_Driven.quantile(0.995))
)
```

```
[37]: plt.figure(figsize=(10, 2))
plt.boxplot(df_train["Kilometers_Driven"], vert=False)
plt.xlabel("Kilometers Driven")
plt.title("Box Plot of Kilometers Driven")
plt.tight_layout()
plt.show()
```



```
[38]: class KilometersDrivenClipper(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        kilometers_driven_col: str = "Kilometers_Driven",
        clipping_quantile: float = 0.995,
    ):
        self.kilometers_driven_col = kilometers_driven_col
        self.clipping_quantile = clipping_quantile

    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "KilometersDrivenClipper":
        return self

    def transform(self, X: pd.DataFrame):
        X = X.copy()
        X[self.kilometers_driven_col] = X[self.kilometers_driven_col].clip(
            upper=int(X[self.kilometers_driven_col].quantile(self.
            ↪clipping_quantile)))
        )
        return X
```

1. Kilometers_Driven
1. Kilometers_Driven Outliers clipped

```
[39]: kilometers_driven_outlier_clip_exp = Experiment(
    ExperimentConfig(
        name="clipping-extreme-Kilometers_Driven",
        pipeline=Pipeline(
            [
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

kilometers_driven_outlier_clip_exp.run(X_train, y_train, X_test, ↪
    baseline_exp_result);
```

[Experiment: clipping-extreme-Kilometers_Driven]
Cross-validating (5-folds)...
CV score: 0.1596 ± 0.0165
-0.0010 +0.0002 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/clipping-extreme-

```
Kilometers_Driven.csv
Experiment complete
```

The performance increased slightly but also increased std slightly. The effect is negligible but still good to have for numerical stability.

```
[40]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

current_best_exp.run(X_train, y_train, None, baseline_exp_result);

[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1485 ± 0.0167
-0.0121 +0.0004 compared to baseline (Negative is better)
Training on full training set...
Experiment complete
```

Fuel_Type

```
[41]: df_train.Fuel_Type.value_counts()
```

```
[41]: Fuel_Type
Diesel      2372
Petrol     2008
CNG         42
LPG          6
Electric      0
Name: count, dtype: int64
```

```
[42]: df_test.Fuel_Type.value_counts()
```

```
[42]: Fuel_Type
Diesel      790
Petrol      685
CNG         13
LPG          3
Electric     0
Name: count, dtype: int64
```

Fuel_Type has low cardinality but CNG, LPG, and Electric has extremely low datapoints. Grouping CNG, LPG, and Electric to Other could mitigate this problem.

```
[43]: class FuelTypeGrouper(TransformerMixin, BaseEstimator):
    def __init__(self, fuel_type_col: str = "Fuel_Type"):
        self.fuel_type_col = fuel_type_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "FuelTypeGrouper":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.fuel_type_col] = (
            X[self.fuel_type_col]
            .astype("object")
            .replace({"CNG": "Other", "LPG": "Other", "Electric": "Other"})
            .astype("category")
        )
        return X
```

```
[44]: fuel_type_grouping_exp = Experiment(
    ExperimentConfig(
        name="grouping-infrequent-fuel-type",
        pipeline=Pipeline(
            [
                ("group_infrequent_fuel_type", FuelTypeGrouper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

fuel_type_grouping_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```
[Experiment: grouping-infrequent-fuel-type]
Cross-validating (5-folds)...
CV score: 0.1601 ± 0.0150
-0.0005 -0.0013 compared to baseline (Negative is better)
Training on full training set...
```

```
Creating submission on test set...
Submission created: artifacts/experiment-results/grouping-infrequent-fuel-
type.csv
Experiment complete
```

Due to low cardinality, (Petrol / Diesel / Other), I do not feel the need to engineer it further (e.g. target encoding).

```
[45]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("group_infrequent_fuel_type", FuelTypeGrouper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

current_best_exp.run(X_train, y_train, None, baseline_exp_result);

[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1489 ± 0.0151
-0.0117 -0.0012 compared to baseline (Negative is better)
Training on full training set...
Experiment complete
```

Transmission

```
[46]: df_train.Transmission.value_counts()
```

```
[46]: Transmission
      Manual      3162
      Automatic   1266
      Name: count, dtype: int64
```

Transmission has two categories and well balanced. There is no need to engineer this feature alone further.

Owner Type

```
[47]: df_train.Owner_Type.cat.categories
```

```
[47]: Index(['First', 'Second', 'Third', 'Fourth & Above'], dtype='object')
```

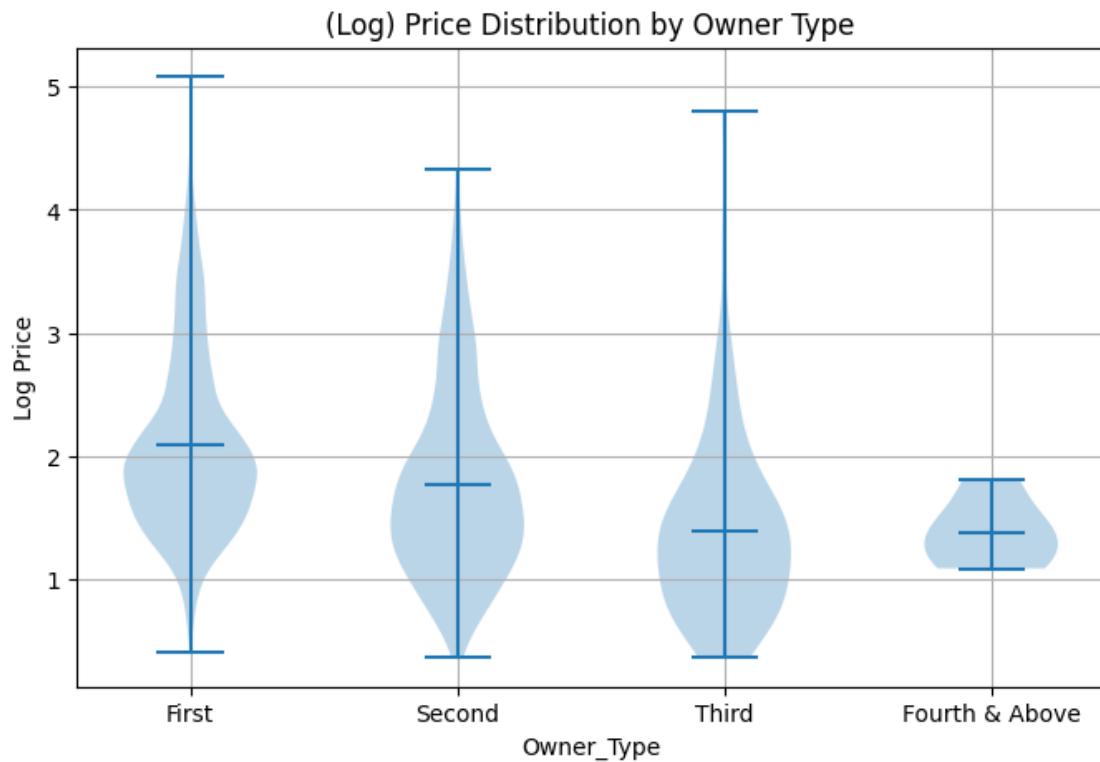
```
[48]: df_train.Owner_Type.value_counts()
```

```
[48]: Owner_Type
```

```
First           3617
Second          715
Third           91
Fourth & Above      5
Name: count, dtype: int64
```

Based on common knowledge, Owner_Type might be one of the most significant predictor of Price.

```
[49]: categories = list(df_train.Owner_Type.cat.categories)
plt.figure(figsize=(8, 5))
plt.violinplot(
    [
        np.log1p(df_train[df_train.Owner_Type == category]["Price"])
        for category in categories
    ],
    showmeans=True,
)
plt.xticks(ticks=range(1, len(categories) + 1), labels=categories)
plt.xlabel("Owner_Type")
plt.ylabel("Log Price")
plt.title("(Log) Price Distribution by Owner Type")
plt.grid(True)
plt.show()
```

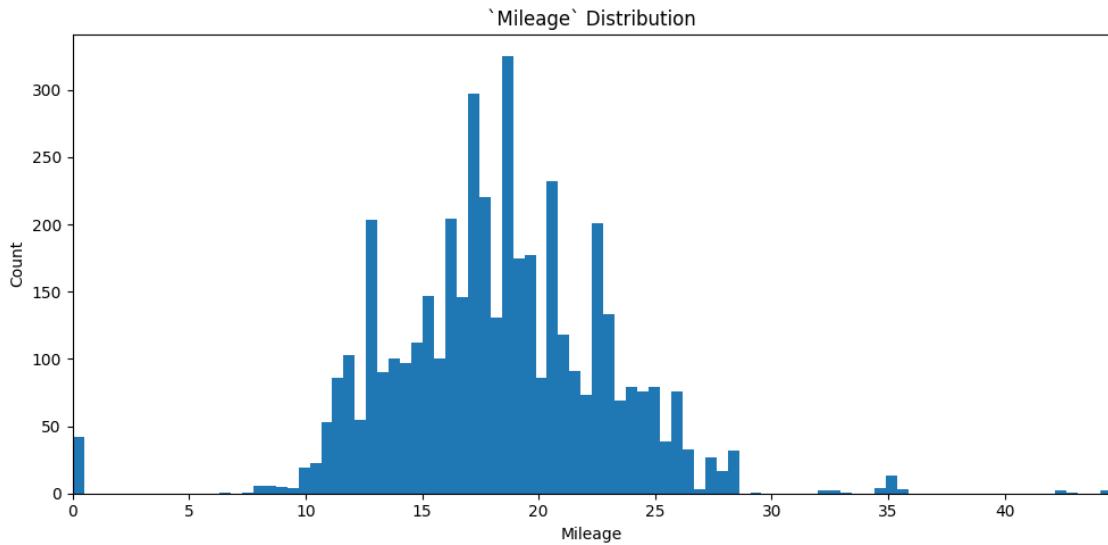


Clearly, as `Owner_Type` increases, `Price` drops monotonically. (`Owner_Type` is already encoded as ordinal category in preprocessing). It is good enough for now.

Mileage

```
[50]: plt.figure(figsize=(10, 5))
plt.hist(df_train.Mileage, bins=100)
limit = np.percentile(df_train.Mileage, 99.9)
plt.xlim(0, limit)
plt.xlabel("Mileage")
plt.ylabel("Count")
plt.title("`Mileage` Distribution")

plt.tight_layout()
plt.show()
```



```
[51]: with pd.option_context("display.max_rows", None):
    display(df_train.Mileage.value_counts().sort_index())
```

Mileage	Count
0.0000	42
6.4000	1
7.5000	1
7.8100	1
7.9400	2
8.0000	1
8.2000	2
8.4500	1
8.5000	1
8.6000	2
8.7000	2
9.0000	5
9.4300	1
9.5000	2
9.5200	1
9.7400	2
9.8000	3
9.9000	2
10.0000	5
10.1000	4
10.1300	3
10.2000	3
10.3700	2
10.4000	2
10.5000	16

10.8000	7
10.9000	6
10.9100	10
10.9300	5
10.9800	1
11.0000	14
11.0500	2
11.0700	1
11.1000	7
11.1800	6
11.2000	4
11.2500	2
11.3000	7
11.3300	5
11.3600	18
11.4000	4
11.4900	1
11.5000	31
11.5600	1
11.5700	6
11.6200	1
11.6800	4
11.7000	16
11.7200	2
11.7400	10
11.7900	4
11.8000	6
11.9000	3
12.0000	11
12.0500	27
12.0700	18
12.1000	2
12.1900	1
12.3000	4
12.3500	1
12.3900	5
12.4000	8
12.5000	3
12.5100	2
12.5500	26
12.6000	5
12.6200	3
12.6300	2
12.6500	2
12.7000	16
12.8000	49
12.8100	2
12.8300	2

12.8500	1
12.9000	17
12.9500	1
12.9700	1
12.9800	1
12.9900	46
13.0000	52
13.0100	7
13.0600	1
13.1000	14
13.1400	3
13.1700	1
13.2000	15
13.2200	4
13.2400	4
13.2900	1
13.3300	2
13.4000	10
13.4400	1
13.4900	2
13.5000	31
13.5300	2
13.6000	13
13.6800	15
13.7000	22
13.7300	3
13.8000	15
13.9000	4
13.9300	6
14.0000	20
14.0200	2
14.0700	1
14.1000	2
14.1600	14
14.2000	4
14.2100	12
14.2400	6
14.2800	16
14.3000	8
14.3300	1
14.3900	1
14.4000	10
14.4200	1
14.4700	1
14.4900	5
14.5300	15
14.5900	1
14.6000	2

14.6200	1
14.6600	4
14.6700	6
14.6900	6
14.7000	9
14.7400	11
14.7500	4
14.8000	13
14.8100	1
14.8400	17
14.9400	4
14.9500	4
15.0000	29
15.0400	15
15.0600	2
15.1000	46
15.1100	1
15.1500	2
15.1700	3
15.2000	5
15.2600	15
15.2900	14
15.3000	14
15.4000	15
15.4100	1
15.5000	14
15.6000	18
15.6300	4
15.6400	5
15.6800	4
15.7000	5
15.7300	15
15.7400	3
15.8000	25
15.8500	1
15.8700	2
15.9000	5
15.9600	7
15.9700	6
16.0000	63
16.0200	12
16.0500	1
16.0700	13
16.0900	7
16.1000	22
16.1200	1
16.2000	18
16.2500	2

16.3000	4
16.3600	12
16.3800	1
16.4000	5
16.4600	2
16.4700	41
16.5000	10
16.5100	1
16.5200	4
16.5500	23
16.6000	7
16.7300	6
16.7700	5
16.7800	7
16.8000	42
16.8200	4
16.9000	6
16.9300	2
16.9500	24
16.9600	5
16.9800	2
17.0000	126
17.0100	29
17.0500	15
17.0900	1
17.1000	20
17.1100	21
17.1600	1
17.1900	4
17.2000	7
17.2100	12
17.2400	1
17.3000	24
17.3200	3
17.4000	27
17.4300	2
17.4400	2
17.4500	2
17.5000	31
17.5400	2
17.5560	1
17.5700	14
17.6000	13
17.6700	3
17.6800	22
17.7000	8
17.7100	6
17.7200	2

17.8000	62
17.8400	1
17.8500	2
17.8800	3
17.9000	34
17.9200	14
17.9700	2
18.0000	53
18.0600	5
18.1000	7
18.1200	9
18.1500	4
18.1600	14
18.1900	3
18.2000	16
18.2300	1
18.2500	4
18.3000	5
18.3300	3
18.4000	5
18.4400	3
18.4800	5
18.4900	6
18.5000	43
18.5100	1
18.5300	3
18.5600	4
18.5900	2
18.6000	77
18.7000	24
18.7800	1
18.8000	2
18.8600	1
18.8800	20
18.9000	133
19.0000	30
19.0100	29
19.0800	9
19.0900	3
19.1000	27
19.1200	6
19.1500	3
19.1600	3
19.2000	7
19.2700	23
19.3000	27
19.3249	1
19.3300	5

19.3400	2
19.4000	31
19.4400	1
19.4900	1
19.5000	13
19.5900	3
19.6000	9
19.6400	8
19.6700	24
19.6900	2
19.7000	33
19.7100	3
19.7200	1
19.8100	33
19.8300	1
19.8700	14
19.9100	1
20.0000	48
20.0830	5
20.1400	19
20.3000	5
20.3400	8
20.3600	64
20.3700	3
20.3800	3
20.4000	21
20.4500	13
20.4600	4
20.5000	12
20.5100	19
20.5400	27
20.5800	3
20.6300	1
20.6400	4
20.6500	1
20.6800	6
20.7000	6
20.7300	17
20.7700	28
20.8500	7
20.8600	1
20.8900	3
20.9200	13
21.0000	1
21.0200	3
21.0300	2
21.1000	65
21.1200	5

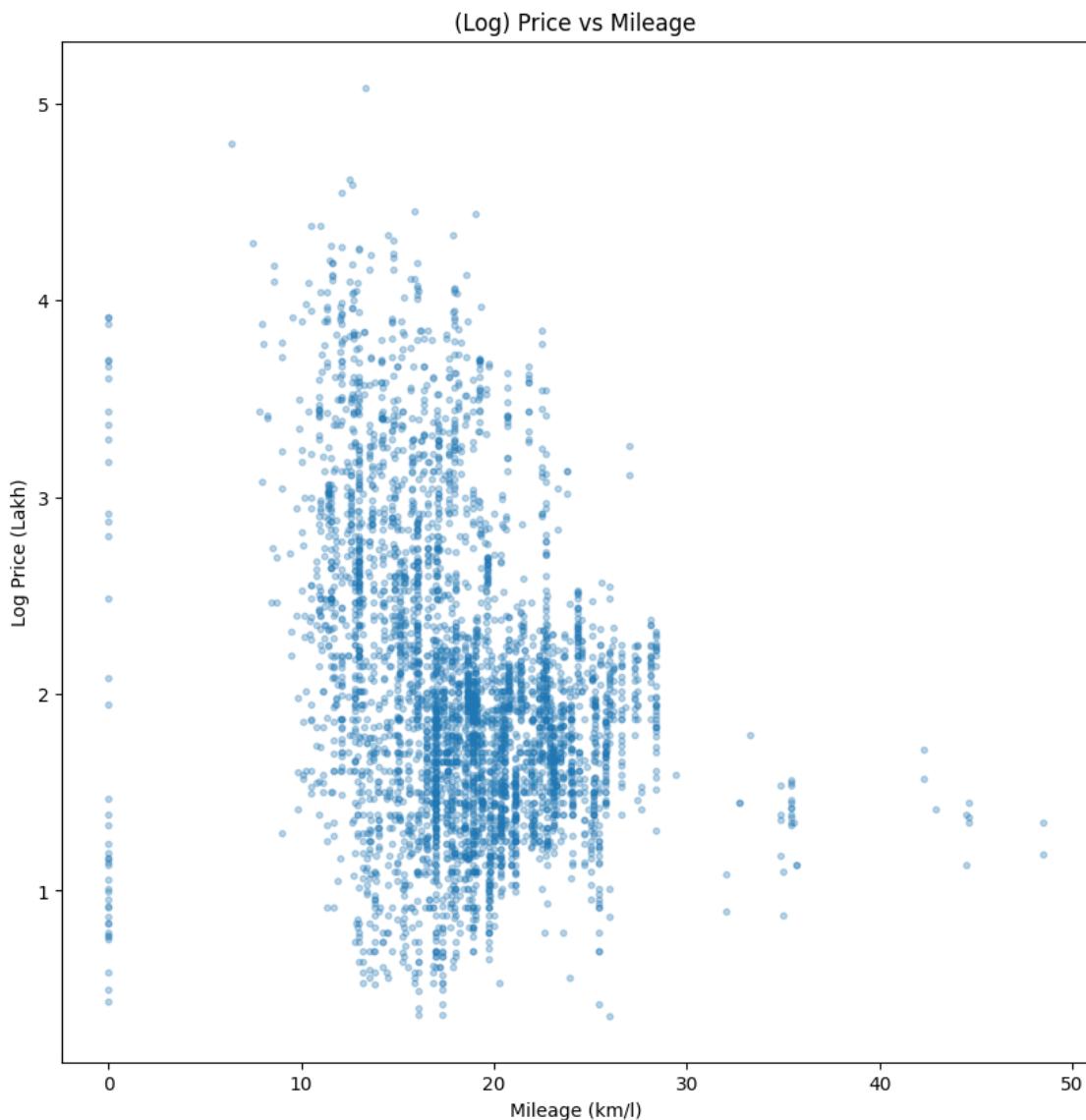
21.1300	1
21.1400	3
21.1900	4
21.2000	1
21.2100	3
21.2700	6
21.3800	2
21.4000	29
21.4300	9
21.5000	17
21.5600	2
21.6400	11
21.6600	5
21.7000	4
21.7600	10
21.7900	1
21.8000	1
21.9000	27
22.0000	18
22.0700	26
22.1000	2
22.3000	7
22.3200	39
22.4800	11
22.5000	19
22.5400	25
22.6100	1
22.6900	19
22.7000	49
22.7400	21
22.7700	10
22.9000	36
22.9500	6
23.0000	22
23.0100	1
23.0800	12
23.1000	39
23.2000	17
23.2750	2
23.3000	2
23.4000	28
23.5000	4
23.5700	1
23.5900	27
23.6500	5
23.8000	3
23.8400	7
23.9000	7

24.0000	35
24.0400	1
24.0700	23
24.2000	3
24.3000	35
24.4000	16
24.5000	3
24.5200	8
24.7000	14
24.8000	2
24.8825	1
25.0000	11
25.1000	14
25.1700	25
25.2000	26
25.3200	4
25.4000	13
25.4400	13
25.4700	7
25.6000	2
25.8000	39
25.8300	8
26.0000	25
26.1000	4
26.2100	9
26.5900	23
26.6000	1
26.8000	1
27.0300	2
27.2800	1
27.3000	13
27.3900	8
27.4000	2
27.6200	3
28.0900	17
28.4000	32
29.3930	1
32.0050	2
32.7180	2
33.2500	1
34.8460	4
34.9790	2
35.3780	11
35.5200	1
35.6839	2
42.2807	2
42.9058	1
44.4752	2

```
44.6082      3  
48.4700      2  
Name: count, dtype: int64
```

It is bell shaped as expected. However, the regular peaks hints that there was some bias towards round numbers.

```
[52]: plt.figure(figsize=(10, 10))  
plt.scatter(df_train.Mileage, np.log1p(df_train.Price), alpha=0.3, s=10)  
plt.xlabel("Mileage (km/l)")  
plt.ylabel("Log Price (Lakh)")  
plt.title("(Log) Price vs Mileage")  
plt.show()
```



Mileage alone does not show clear relationship with Price.

Cleaning up unrealistic outliers would stabilize the model.

```
[53]: class MileageClipper(TransformerMixin, BaseEstimator):
    def __init__(self,
                 mileage_col: str = "Mileage",
                 clipping_quantile: float = 0.995,
                 ):
        self.mileage_col = mileage_col
        self.clipping_quantile = clipping_quantile

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series] = None) -> "MileageClipper":
        return self

    def transform(self, X: pd.DataFrame):
        X = X.copy()
        X[self.mileage_col] = X[self.mileage_col].clip(
            upper=int(X[self.mileage_col].quantile(self.clipping_quantile)))
        )
        return X
```

```
[54]: mileage_outlier_clip_exp = Experiment(
    ExperimentConfig(
        name="clipping-extreme-mileage",
        pipeline=Pipeline(
            [
                ("mileage_clip_outliers", MileageClipper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

mileage_outlier_clip_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```
[Experiment: clipping-extreme-mileage]
Cross-validating (5-folds)...
CV score: 0.1578 ± 0.0153
-0.0028 -0.0010 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/clipping-extreme-mileage.csv
Experiment complete
```

```
[55]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("group_infrequent_fuel_type", FuelTypeGrouper()),
                ("mileage_clip_outliers", MileageClipper()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

current_best_exp.run(X_train, y_train, None, baseline_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1484 ± 0.0143
-0.0123 -0.0020 compared to baseline (Negative is better)
Training on full training set...
Experiment complete
```

Engine

```
[56]: df_train.Engine.describe()
```

```
[56]: count      4428.0
mean       1618.2771
std        595.245047
min         624.0
25%        1198.0
50%        1493.0
75%        1968.0
max        5998.0
Name: Engine, dtype: Float64
```

Engine feature seems reasonable enough.

Power

```
[57]: df_train.Power.describe()
```

```
[57]: count    4336.000000
      mean     113.171817
      std      53.875993
      min     34.200000
      25%    75.000000
      50%    94.000000
      75%   138.100000
      max    560.000000
Name: Power, dtype: float64
```

`Power` feature seems reasonable enough.

However, as explored in `preprocessing.ipynb`, the percentage of missing values in `Power` was non-negligible.

```
[58]: df_train[df_train.Power.isna()]
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	\	
ID							
2CM572	Fiat Petra	Pune	2005	120000	Petrol		
4J1SFY	Mercedes-Benz E-Class	Pune	2001	121000	Diesel		
LHXSLV	Maruti Swift	Hyderabad	2014	81609	Diesel		
R7UPR3	Fiat Siena	Jaipur	2001	70000	Petrol		
KZI5XI	Skoda Laura	Pune	2010	85000	Petrol		
...		
Z3GW3N	Hyundai Santro	Hyderabad	2006	74483	Petrol		
M49PV9	Hyundai Santro	Mumbai	2005	102000	Petrol		
9BPQA2	Hyundai Santro	Pune	2005	100000	CNG		
AOLNGH	Hyundai Santro	Jaipur	2004	200000	Petrol		
MDPHOT	Hyundai Santro	Kochi	2007	58815	Petrol		
	Transmission	Owner_Type	Mileage	Engine	Power	\	
ID							
2CM572	Manual	Second	15.50	1242	Nan	Others	5
4J1SFY	Manual	First	15.00	2148	Nan	White	5
LHXSLV	Manual	First	17.80	1248	Nan	Others	5
R7UPR3	Manual	Third	0.00	1242	Nan	White	5
KZI5XI	Manual	First	17.50	1798	Nan	Black/Silver	5
...	
Z3GW3N	Automatic	First	0.00	999	Nan	Black/Silver	5
M49PV9	Manual	Second	17.00	1086	Nan	White	5
9BPQA2	Manual	Third	22.61	1086	Nan	White	5
AOLNGH	Manual	First	0.00	1086	Nan	Black/Silver	5
MDPHOT	Manual	First	17.00	1086	Nan	White	5
	Doors	New_Price	Price				
ID							
2CM572	4	Nan	0.85				

```

4J1SFY      4      NaN  5.00
LHXSLV      4      NaN  5.55
R7UPR3      4      NaN  0.55
KZI5XI      4      NaN  2.85
...
Z3GW3N      4      NaN  2.30
M49PV9       4      NaN  0.85
9BPQA2      4      NaN  1.20
AOLNGH      4      NaN  0.80
MDPHOT      4      NaN  1.99

```

[92 rows x 15 columns]

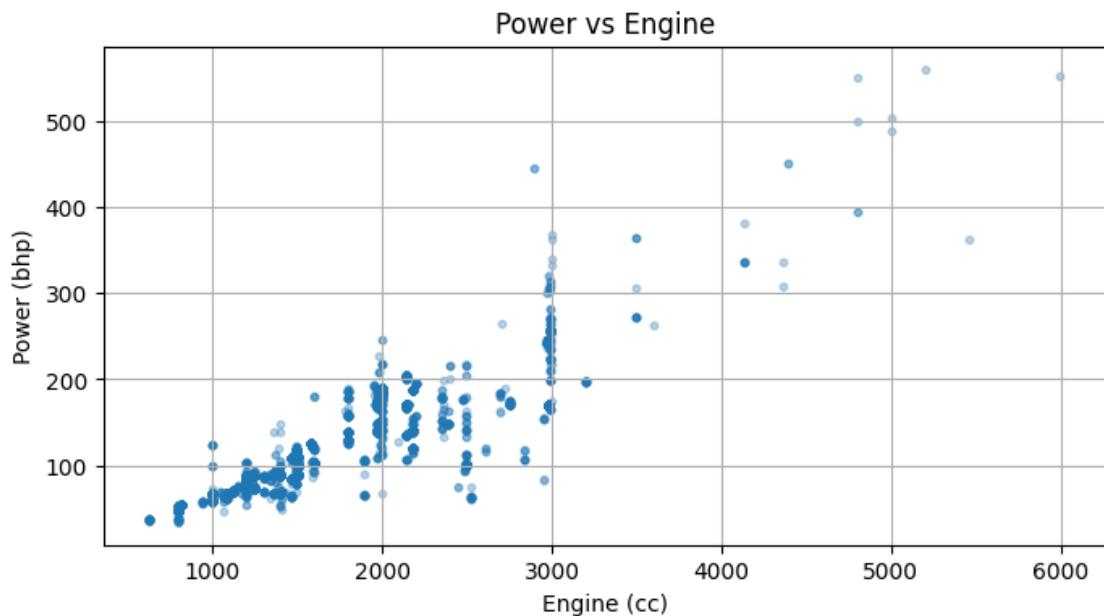
Power can be faithfully imputed exploiting its high relevance to Engine.

```
[59]: print(f"Engine-Power Correlation: {df_train.Engine.corr(df_train.Power):.2f}")
```

Engine-Power Correlation: 0.86

```
[60]: plt.figure(figsize=(8, 4))
plt.scatter(df_train["Engine"], df_train["Power"], alpha=0.3, s=10)

plt.xlabel("Engine (cc)")
plt.ylabel("Power (bhp)")
plt.title("Power vs Engine")
plt.grid(True)
plt.show()
```



```
[61]: class PowerImputer(TransformerMixin, BaseEstimator):
    def __init__(self, engine_col: str = "Engine", power_col="Power", clip_negative: bool = True):
        self.engine_col = engine_col
        self.power_col = power_col
        self.clip_negative = clip_negative

    def fit(self, X: pd.DataFrame, y=Optional[pd.Series]) -> "PowerImputer":
        df = X[[self.engine_col, self.power_col]].dropna()
        engines = df[self.engine_col].astype(float)
        powers = df[self.power_col].astype(float)

        # Linear regression
        cov = ((engines - engines.mean()) * (powers - powers.mean())).sum()
        var = ((engines - engines.mean()) ** 2).sum()
        self.slope = cov / var
        self.intercept = powers.mean() - self.slope * engines.mean()

        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        mask = X[self.power_col].isna()
        has_engine = X[self.engine_col].notna() & mask
        X.loc[has_engine, self.power_col] = (
            self.slope * X.loc[has_engine, self.engine_col] + self.intercept
        )

        if self.clip_negative:
            X[self.power_col] = X[self.power_col].clip(lower=0)

        return X
```

```
[62]: power_imputation_exp = Experiment(
    ExperimentConfig(
        name="power-imputation",
        pipeline=Pipeline(
            [
                ("imput_power", PowerImputer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)
```

```
)
```

```
power_imputation_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```
[Experiment: power-imputation]
Cross-validating (5-folds)...
CV score: 0.1551 ± 0.0159
    -0.0055 -0.0004 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/power-imputation.csv
Experiment complete
```

```
[63]: imputer = PowerImputer()
imputer.fit(X_train)

slope = imputer.slope
intercept = imputer.intercept

df_plot = X_train.copy()
df_plot["imputed_Power"] = imputer.transform(X_train)[imputer.power_col]
df_plot["missing_Power"] = df_plot.Power.isna()

engine_range = np.linspace(df_plot.Engine.min(), df_plot.Engine.max(), 200)
regression_line = slope * engine_range + intercept

observed_mask = ~df_plot.missing_Power

df_observed = df_plot.loc[observed_mask]
residuals = df_observed.Power - (slope * df_observed.Engine + intercept)
sigma = residuals.std()

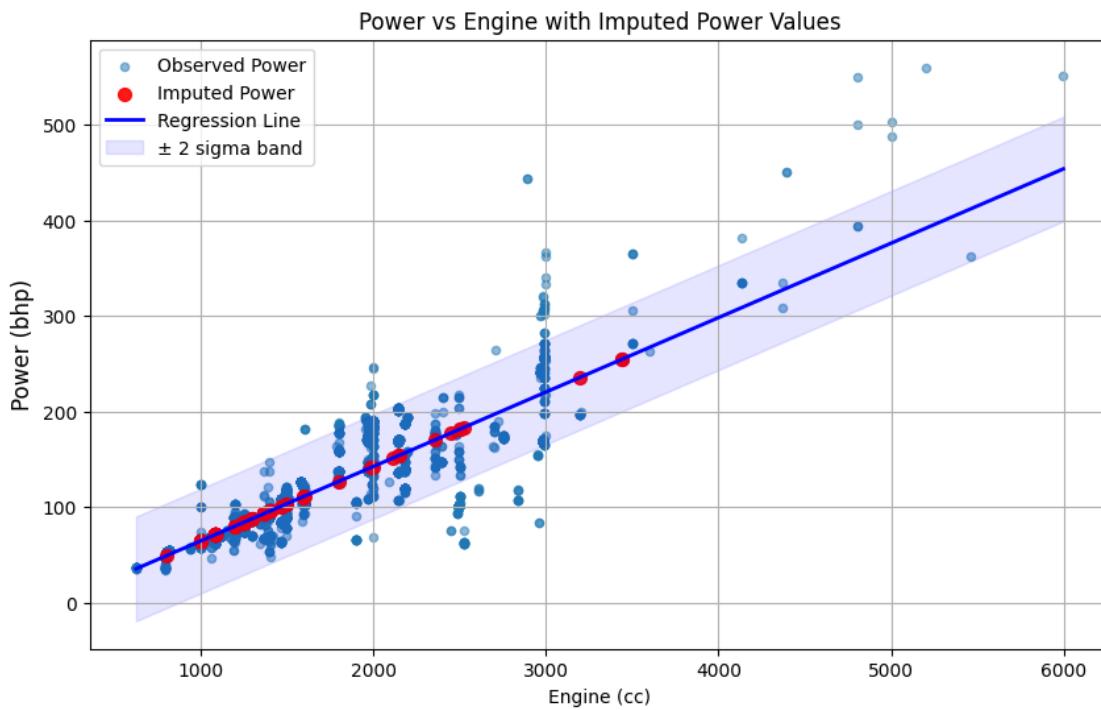
plt.figure(figsize=(10, 6))
plt.scatter(
    df_plot.loc[observed_mask, "Engine"],
    df_plot.loc[observed_mask, "Power"],
    s=20,
    alpha=0.5,
    label="Observed Power",
)
plt.scatter(
    df_plot.loc[df_plot.missing_Power, "Engine"],
    df_plot.loc[df_plot.missing_Power, "imputed_Power"],
    s=50,
    color="red",
    alpha=0.9,
```

```

        label="Imputed Power",
)
plt.plot(
    engine_range, regression_line, color="blue", linewidth=2, label="Regression Line"
)
plt.fill_between(
    engine_range,
    regression_line - 2 * sigma,
    regression_line + 2 * sigma,
    color="blue",
    alpha=0.1,
    label="± 2 sigma band",
)

plt.xlabel("Engine (cc)", fontsize=10)
plt.ylabel("Power (bhp)", fontsize=12)
plt.title("Power vs Engine with Imputed Power Values")
plt.legend()
plt.grid(True)
plt.show()

```



Missing **Power** was effectively imputed with its highly linear correlation with **Engine**.

Combining all feature engineering so far.

```
[64]: df_train.columns
```

```
[64]: Index(['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Colour',
       'Seats', 'Doors', 'New_Price', 'Price'],
      dtype='object')
```

```
[65]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("group_infrequent_fuel_type", FuelTypeGrouper()),
                ("mileage_clip_outliers", MileageClipper()),
                ("imput_power", PowerImputer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    ),
)

current_best_exp.run(X_train, y_train, None, baseline_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1466 ± 0.0165
-0.0140 +0.0002 compared to baseline (Negative is better)
Training on full training set...
Experiment complete
```

Colour

```
[66]: df_train.Colour.describe()
```

```
[66]: count      4428
unique        3
top        White
freq       1539
Name: Colour, dtype: object
```

```
[67]: df_train.Colour.value_counts()
```

```
[67]: Colour
White           1539
Others          1519
Black/Silver    1370
Name: count, dtype: int64
```

Colour is already well balanced.

0.0.3 Seats

```
[68]: df_train.Seats.value_counts()
```

```
[68]: Seats
5      3736
7      503
8      83
4      69
6      21
2      12
10     2
9      2
Name: count, dtype: Int64
```

Binning could help for those rare seat counts. However, the binning should not break the ordinality.

```
[69]: class SeatsBinner(TransformerMixin, BaseEstimator):
    def __init__(self, seats_col: str = "Seats"):
        self.seats_col = seats_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "SeatsBinner":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        seats = X[self.seats_col]

        X[self.seats_col] = seats.replace(
            {
                2: 4,
                4: 4,
                # Small
                5: 5,
                # Standard
                6: 7,
                7: 7,
                8: 7,
```

```

        # Large
        9: 9,
        10: 9,
        # Van
    }
)

return X

```

```
[70]: seats_binning_exp = Experiment(
    ExperimentConfig(
        name="bin-seats",
        pipeline=Pipeline(
            [
                ("bin_seats", SeatsBinner()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

seats_binning_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

[Experiment: bin-seats]
Cross-validating (5-folds)...
CV score: 0.1581 ± 0.0154
-0.0025 -0.0009 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/bin-seats.csv
Experiment complete

Binning Seats had positive effect on the prediction score.

```
[71]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
            ]
        )
)
```

```

        ("group_infrequent_fuel_type", FuelTypeGrouper()),
        ("mileage_clip_outliers", MileageClipper()),
        ("imput_power", PowerImputer()),
        ("bin_seats", SeatsBinner()),
        ("category_encode", CategoricalEncoder()),
        ("model", XGBRegressor()),
    ]
),
)
)

current_best_exp.run(X_train, y_train, None, baseline_exp_result);

```

[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1479 ± 0.0159
-0.0127 -0.0004 compared to baseline (Negative is better)
Training on full training set...
Experiment complete

Doors

[72]: df_train.Doors.value_counts()

[72]: Doors

4	3884
5	532
2	12

Name: count, dtype: Int64

There are very small number of cars with 2 doors. However, I think it should not be binned, since usually cars with 2 doors are very expensive sports car, and it can imply a lot, even though it has very small sample size.

[73]: df_train[df_train.Doors == 2][["Name", "Engine", "Power", "Seats", "Doors"]]

[73]:

ID	Name	Engine	Power	Seats	Doors
UFNCV8	BMW Z4	2979	306.00	2	2
ICAVC1	Jaguar F	5000	488.10	2	2
YDPHR8	Smart Fortwo	799	NaN	2	2
RS5FNO	Audi A4	3197	NaN	2	2
X5JJRY	Porsche Boxster	2706	265.00	2	2
2OAGHR	Porsche Cayman	3436	NaN	2	2
ORNZ40	Porsche Cayman	3436	NaN	2	2
W15E5A	Mercedes-Benz SLK-Class	3498	306.00	2	2
NF6IF1	BMW Z4	2979	306.00	2	2
RGDCTT	Mercedes-Benz SLC	2996	362.07	2	2

IJ96ZM	Audi TT	1984	207.80	2	2
PYZN3W	Lamborghini Gallardo	5204	560.00	2	2

New_Price

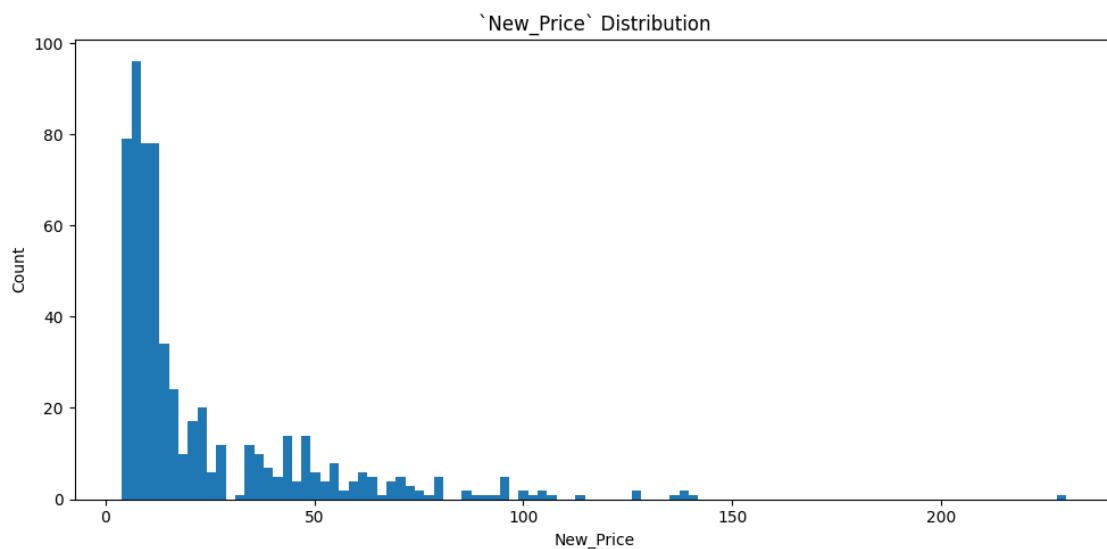
```
[74]: print(
    f"# of missing New_Price: {len(df_train[df_train.New_Price.isna()])} out of {len(df_train)}"
)
```

of missing New_Price: 3827 out of 4428

```
[75]: observed_new_prices = df_train[df_train.New_Price.notna()].New_Price

plt.figure(figsize=(10, 5))
plt.hist(observed_new_prices, bins=100)
plt.xlabel("New_Price")
plt.ylabel("Count")
plt.title("`New_Price` Distribution")

plt.tight_layout()
plt.show()
```



Due to its high sparsity, it would be dangerous to impute naively. Instead, labeling missing New_Price might give non-trivial information from the pattern of missing New_Price. Additionally, log transformation could help the heavy skew in its distribution.

```
[76]: observed_new_prices = df_train[df_train.New_Price.notna()].New_Price

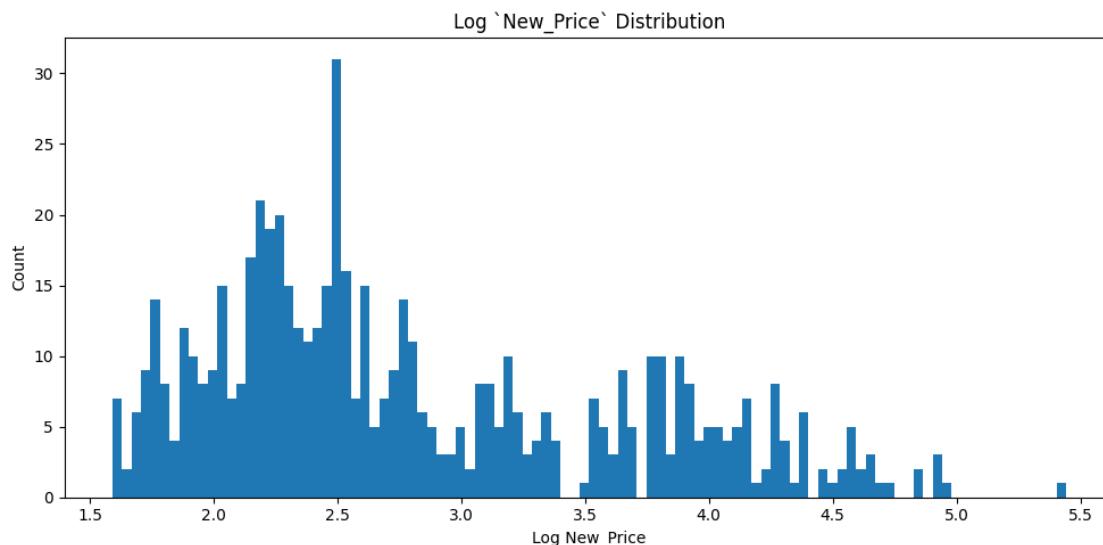
plt.figure(figsize=(10, 5))
```

```

plt.hist(np.log1p(observe_new_prices), bins=100)
plt.xlabel("Log New Price")
plt.ylabel("Count")
plt.title("Log `New_Price` Distribution")

plt.tight_layout()
plt.show()

```



```

[77]: class NewPriceTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, new_price_col: str = "New_Price"):
        self.new_price_col = new_price_col
        self.missing_col = "Missing_" + new_price_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> Union["NewPriceTransformer", None]:
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        X[self.missing_col] = X[self.new_price_col].isna().astype(int)

        X[self.new_price_col] = X[self.new_price_col].apply(np.log1p)

        return X

```

```
[78]: new_price_transform_exp = Experiment(
    ExperimentConfig(

```

```

        name="transform-new-price",
        pipeline=Pipeline(
            [
                ("transform_new_price", NewPriceTransformer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

new_price_transform_exp.run(X_train, y_train, X_test, baseline_exp_result);

```

```

[Experiment: transform-new-price]
Cross-validating (5-folds)...
CV score: 0.1606 ± 0.0163
    +0.0000 +0.0000 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/transform-new-price.csv
Experiment complete

```

It did not yield visible gain. However, I'll keep it since this transformation can be useful in making interaction features. Should be judged later.

```

[79]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                ("extract_brand_model", BrandModelExtractor()),
                (
                    "target_encode",
                    TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
                ),
                ("transform", YearToAgeTransformer()),
                ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
                ("group_infrequent_fuel_type", FuelTypeGrouper()),
                ("mileage_clip_outliers", MileageClipper()),
                ("imput_power", PowerImputer()),
                ("bin_seats", SeatsBinner()),
                ("transform_new_price", NewPriceTransformer()),
                ("category_encode", CategoricalEncoder()),
                ("model", XGBRegressor()),
            ]
        ),
    )
)

```

```
)
```

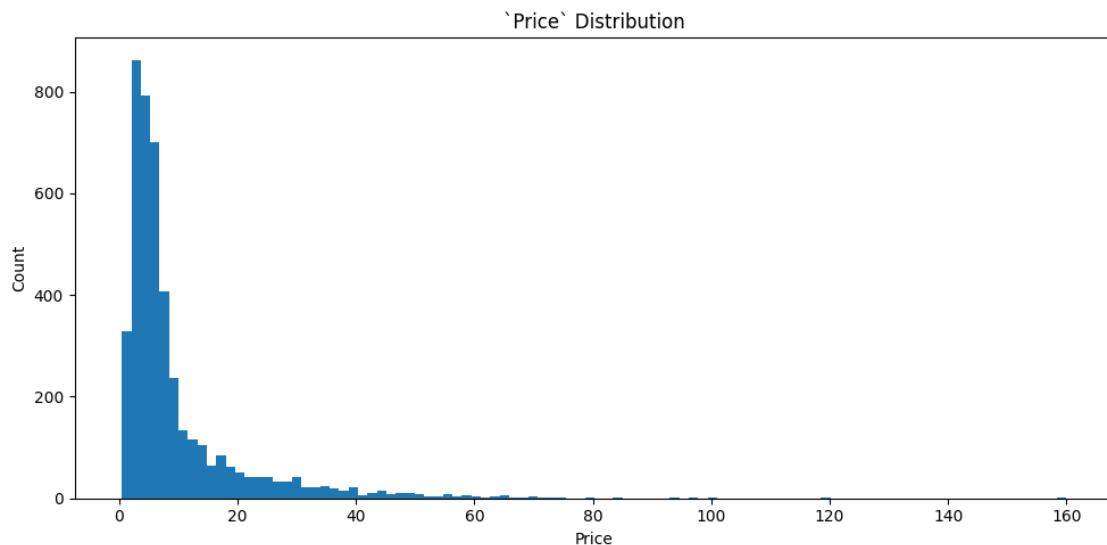
```
current_best_exp.run(X_train, y_train, None, baseline_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1479 ± 0.0159
    -0.0127 -0.0004 compared to baseline (Negative is better)
Training on full training set...
Experiment complete
```

Price

```
[80]: plt.figure(figsize=(10, 5))
plt.hist(df_train.Price, bins=100)
plt.xlabel("Price")
plt.ylabel("Count")
plt.title("`Price` Distribution")

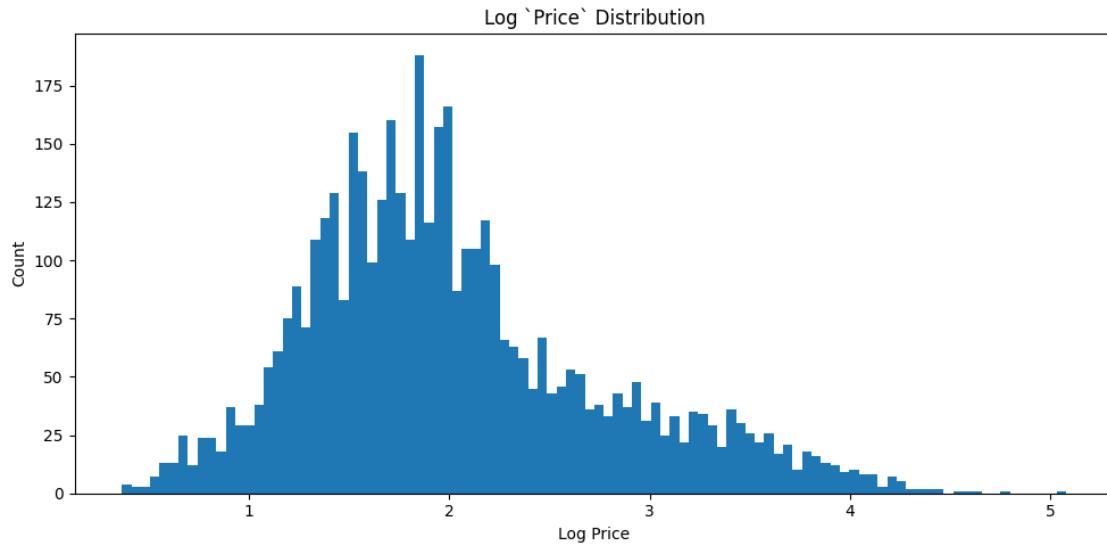
plt.tight_layout()
plt.show()
```



The target feature `Price` is heavily skewed. Log transformation can mediate this.

```
[81]: plt.figure(figsize=(10, 5))
plt.hist(np.log1p(df_train.Price), bins=100)
plt.xlabel("Log Price")
plt.ylabel("Count")
plt.title("Log `Price` Distribution")
```

```
plt.tight_layout()  
plt.show()
```



```
[82]: price_transform_exp = Experiment(  
    ExperimentConfig(  
        name="transform-price",  
        pipeline=Pipeline(  
            [  
                ("category_encode", CategoricalEncoder()),  
                (  
                    "model",  
                    TransformedTargetRegressor(  
                        regressor=XGBRegressor(), func=np.log1p,  
                        inverse_func=np.expm1  
                    ),  
                ),  
            ],  
        ),  
    ),  
)
```

```
price_transform_exp.run(X_train, y_train, X_test, baseline_exp_result);
```

```
[Experiment: transform-price]  
Cross-validating (5-folds)...  
CV score: 0.1456 ± 0.0170  
-0.0150 +0.0007 compared to baseline (Negative is better)  
Training on full training set...
```

```
Creating submission on test set...
Submission created: artifacts/experiment-results/transform-price.csv
Experiment complete
```

Log transformation on Price was very effective.

0.0.4 Combining best transformations for each features

```
[83]: best_single_feature_pipeline = Pipeline(
    [
        ("extract_brand_model", BrandModelExtractor()),
        (
            "target_encode",
            TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
        ),
        ("transform", YearToAgeTransformer()),
        ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
        ("group_infrequent_fuel_type", FuelTypeGrouper()),
        ("mileage_clip_outliers", MileageClipper()),
        ("imput_power", PowerImputer()),
        ("bin_seats", SeatsBinner()),
        ("transform_new_price", NewPriceTransformer()),
        ("category_encode", CategoricalEncoder()),
        (
            "model",
            TransformedTargetRegressor(
                regressor=XGBRegressor(), func=np.log1p, inverse_func=np.expm1
            ),
        ),
    ],
)
```

```
[84]: best_feature_by_feature_exp = Experiment(
    ExperimentConfig(
        name="combine-all-feature-by-feature-engineering",
        pipeline=best_single_feature_pipeline,
    )
)

best_feature_by_feature_exp_result = best_feature_by_feature_exp.run(
    X_train, y_train, X_test, baseline_exp_result
)
```

```
[Experiment: combine-all-feature-by-feature-engineering]
Cross-validating (5-folds)...
CV score: 0.1379 ± 0.0173
```

```

-0.0228 +0.0010 compared to baseline (Negative is better)
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/combine-all-feature-by-feature-
engineering.csv
Experiment complete

```

0.0.5 Interactions

To determine which interactions are worth investigating, I first perform residual diagnostics.

```
[85]: best_single_feature_pipeline.fit(X_train, y_train)

prediction_by_best_single_feature_pipeline = pd.Series(
    best_single_feature_pipeline.predict(X_train), index=y_train.index
)
residuals = y_train - prediction_by_best_single_feature_pipeline

X_train_transformed = pd.DataFrame(
    best_single_feature_pipeline[:-1].fit_transform(X_train, y_train)
)
y_train_transformed = y_train.apply(np.log10)

df_residuals = X_train_transformed.copy()
df_residuals["Residual"] = residuals
df_residuals["Prediction"] = prediction_by_best_single_feature_pipeline
df_residuals["Price"] = y_train

df_residuals
```

```
[85]:   ID      Name  Location Kilometers_Driven Fuel_Type Transmission \
G4XLU0  4.267524  15.550690          59138        0            1
CRSHOS  7.206730  11.252232          81504        0            1
FUJ4X1   6.443593  10.093132          92000        2            1
QMVK6E   5.077432   5.460807          33249        0            1
4SWHFC   5.867476  12.628270          65000        2            1
...
TR7SLB  10.542137  11.252232          ...          ...          ...
QB41QE   6.497267   5.460807          27210        0            1
ODG8N7  33.110594   6.749484          52000        0            0
EV2ZBX   2.542287  10.096643          56000        2            1
J2RCU8  18.250472  12.628270          52000        0            0

Owner_Type  Mileage  Engine  Power  Colour  Seats  Doors  New_Price \

```

ID									
G4XLU0	0	17.00	1405	70.00	1	5	4	NaN	
CRSHOS	0	21.43	1364	87.20	1	5	4	NaN	
FUJ4X1	0	13.80	1299	70.00	1	5	4	NaN	
QMVK6E	0	21.27	1396	88.76	0	5	4	NaN	
4SWHFC	0	17.00	1497	118.00	2	5	4	NaN	
...	
TR7SLB	0	16.00	2179	140.00	2	7	5	NaN	
QB41QE	0	27.30	1498	98.60	1	5	4	NaN	
ODG8N7	0	12.70	2179	187.70	2	5	4	NaN	
EV2ZBX	0	24.70	796	47.30	1	5	4	NaN	
J2RCU8	0	12.00	2987	224.00	0	7	5	NaN	

	Brand	Model	Age	Missing_New_Price	Residual	Prediction	\
ID							
G4XLU0	3.417323	4.267524	7		1 0.021814	2.558186	
CRSHOS	11.417767	7.206730	7		1 -0.088608	6.618608	
FUJ4X1	6.549908	6.443593	13		1 -0.100111	1.350111	
QMVK6E	5.333501	5.077432	8		1 -0.016794	3.266794	
4SWHFC	5.425303	5.867476	9		1 0.109849	5.090151	
...	
TR7SLB	8.075126	10.542137	4		1 0.440084	12.019916	
QB41QE	5.425303	6.497267	4		1 -0.017071	5.867071	
ODG8N7	37.771935	33.110594	5		1 0.467415	39.282585	
EV2ZBX	4.604672	2.542287	7		1 -0.037402	2.137402	
J2RCU8	27.104100	18.250472	6		1 2.699703	46.300297	

Price							
ID							
G4XLU0	2.58						
CRSHOS	6.53						
FUJ4X1	1.25						
QMVK6E	3.25						
4SWHFC	5.20						
...	...						
TR7SLB	12.46						
QB41QE	5.85						
ODG8N7	39.75						
EV2ZBX	2.10						
J2RCU8	49.00						

[4428 rows x 20 columns]

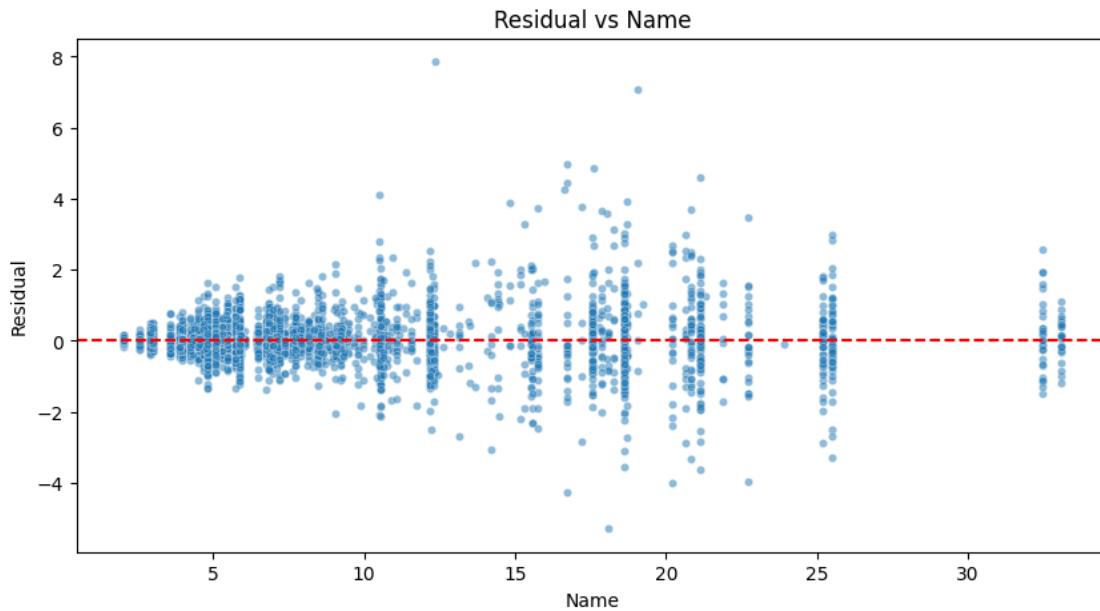
```
[86]: def residual_plot(feature, bins=False):
    plt.figure(figsize=(10, 10 if bins else 5))
    x = df_residuals[feature][df_residuals[feature].notna()]
    if bins:
```

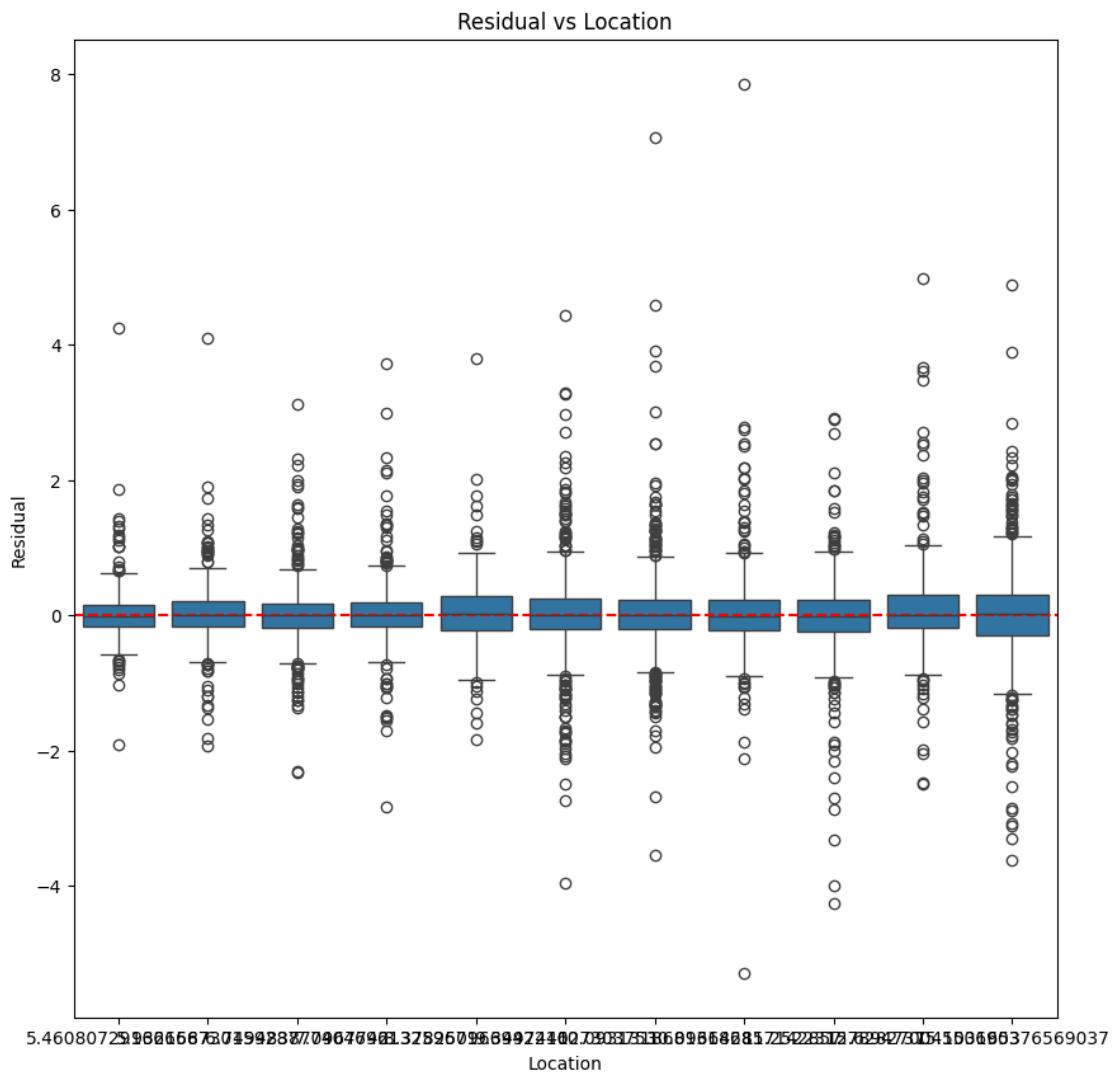
```

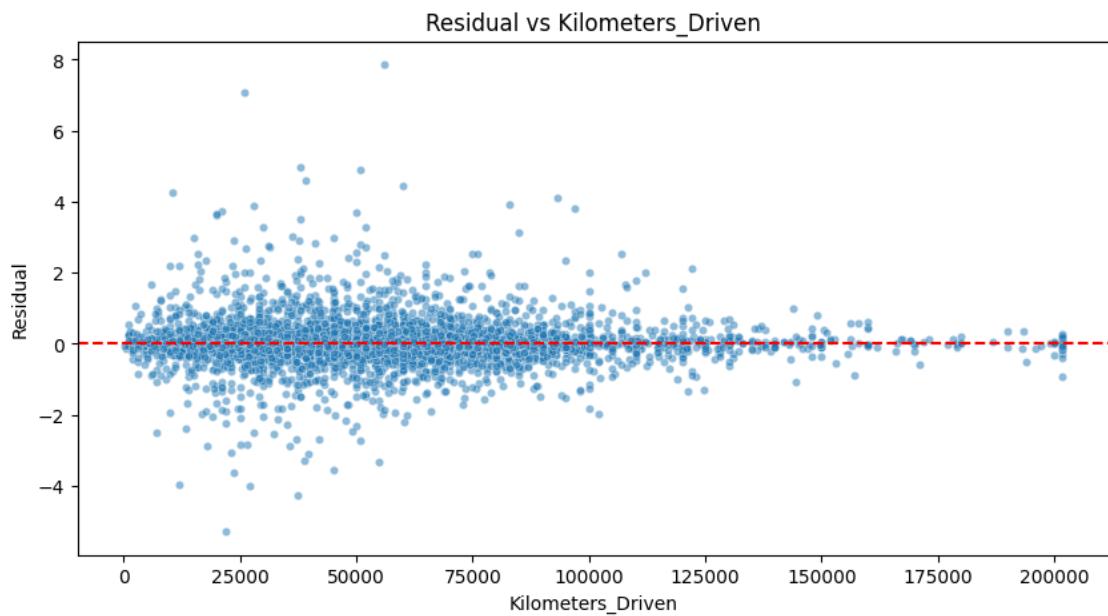
        sns.boxplot(x=x, y=df_residuals["Residual"])
else:
    sns.scatterplot(
        x=x,
        y=df_residuals["Residual"],
        alpha=0.5,
        s=20,
    )
plt.axhline(0, color="red", linestyle="--")
plt.title(f"Residual vs {feature}")
plt.show()

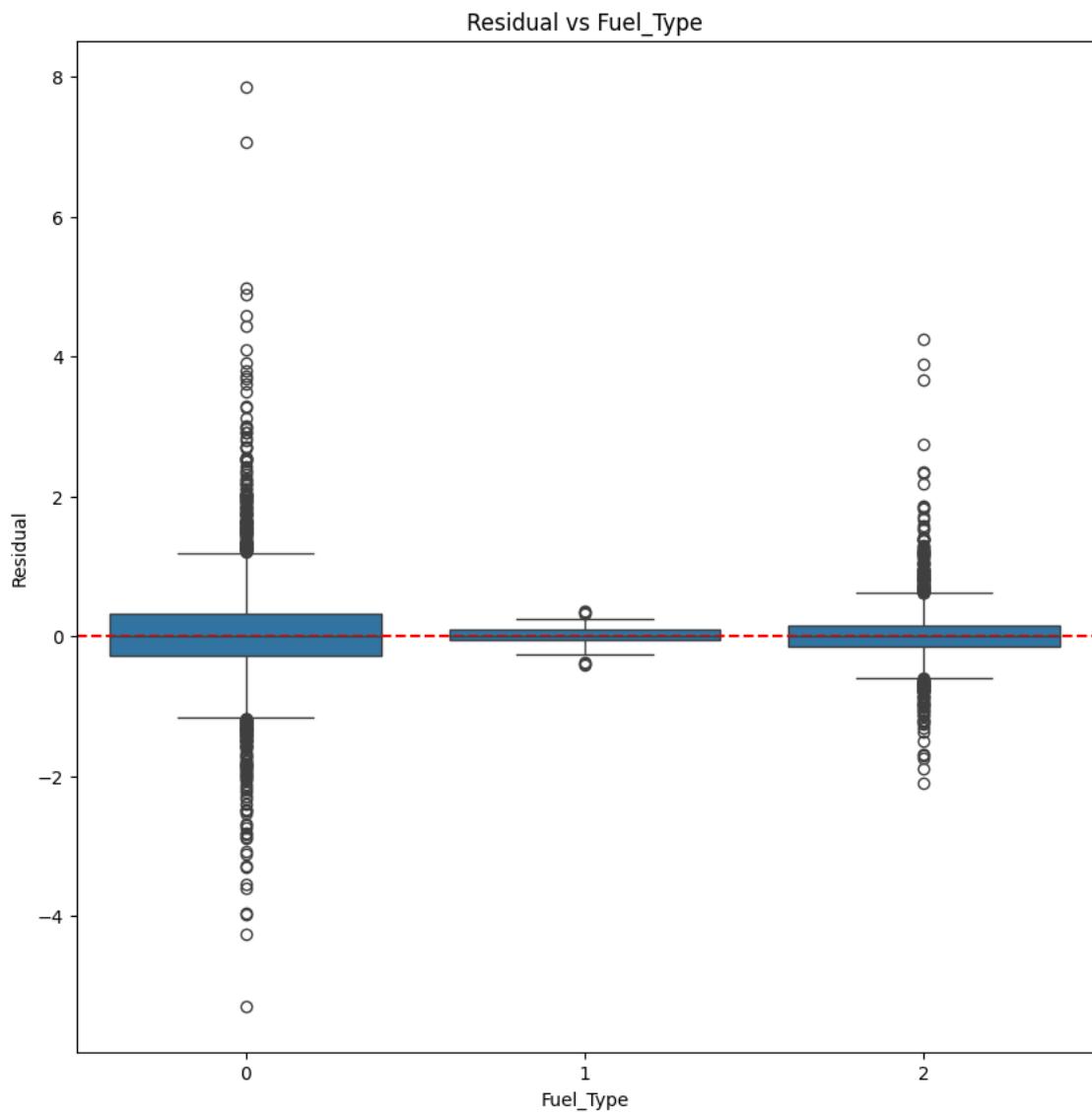
```

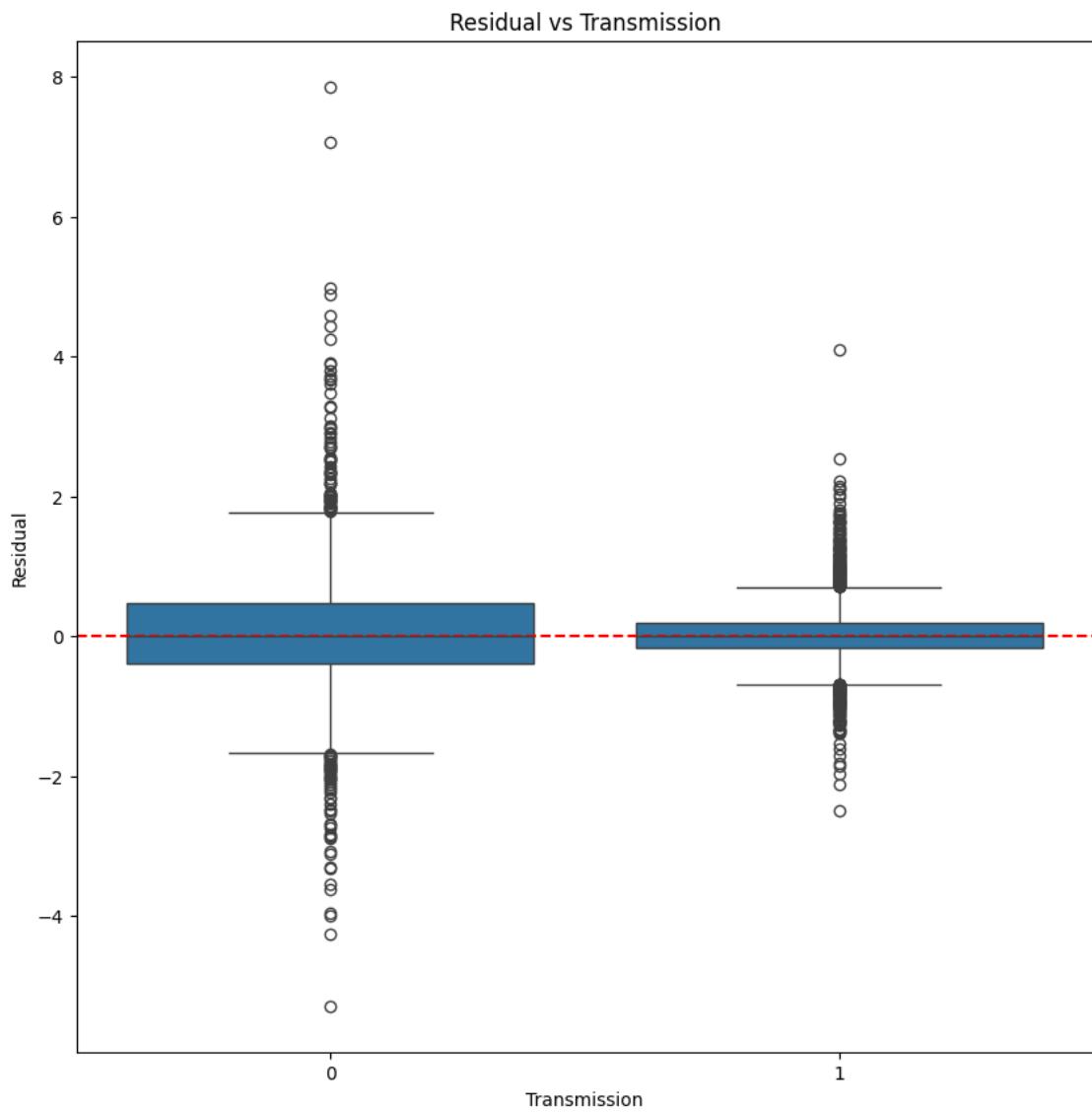
```
[87]: for feature in df_residuals.drop(columns=["Residual", "Price", "Prediction"]).
columns:
    use_box_plot = (
        True
        if len(df_residuals[feature].unique()) < 15
        or df_residuals[feature].dtype.name == "category"
        else False
    )
    residual_plot(feature, bins=use_box_plot)
```

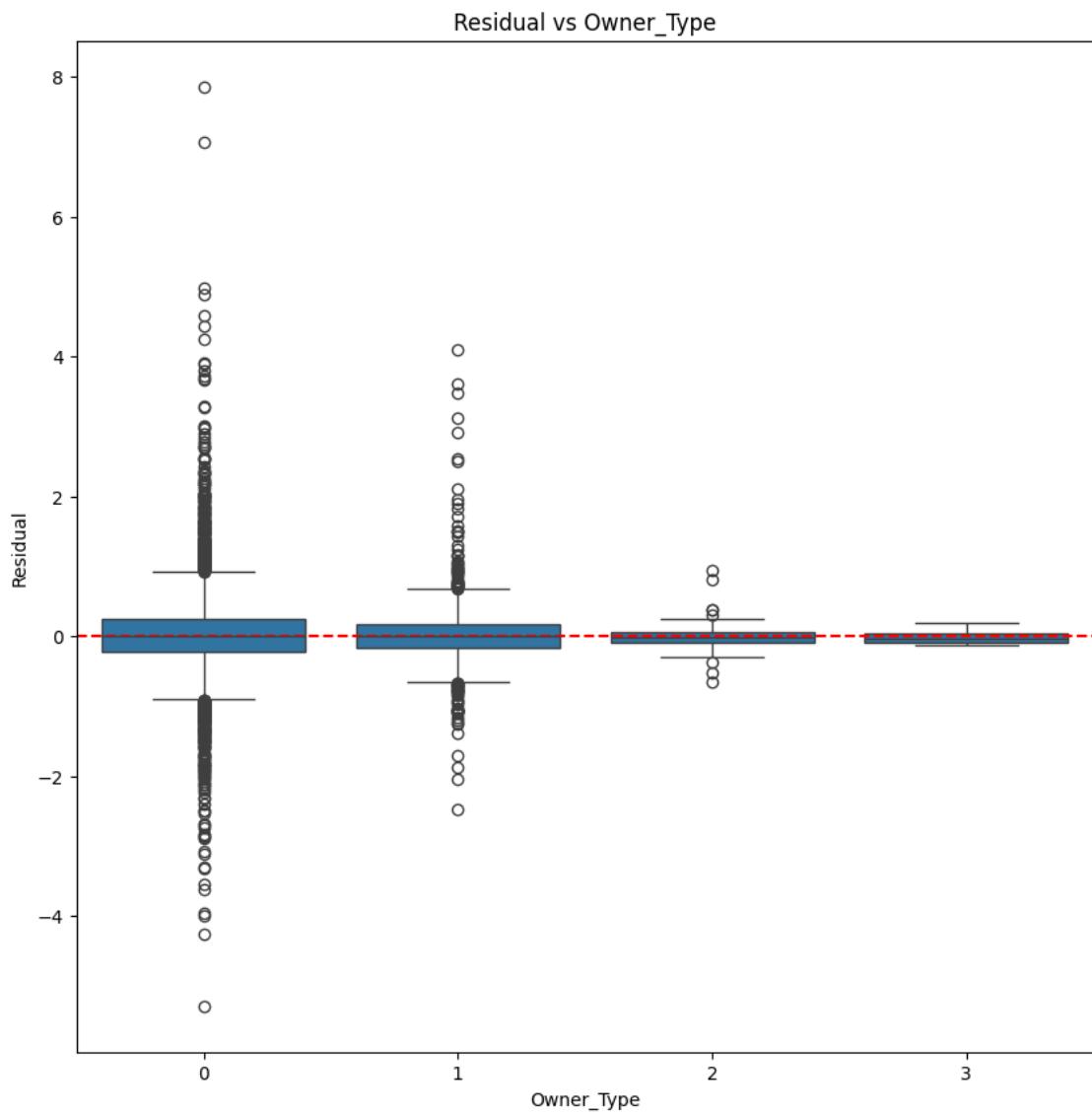




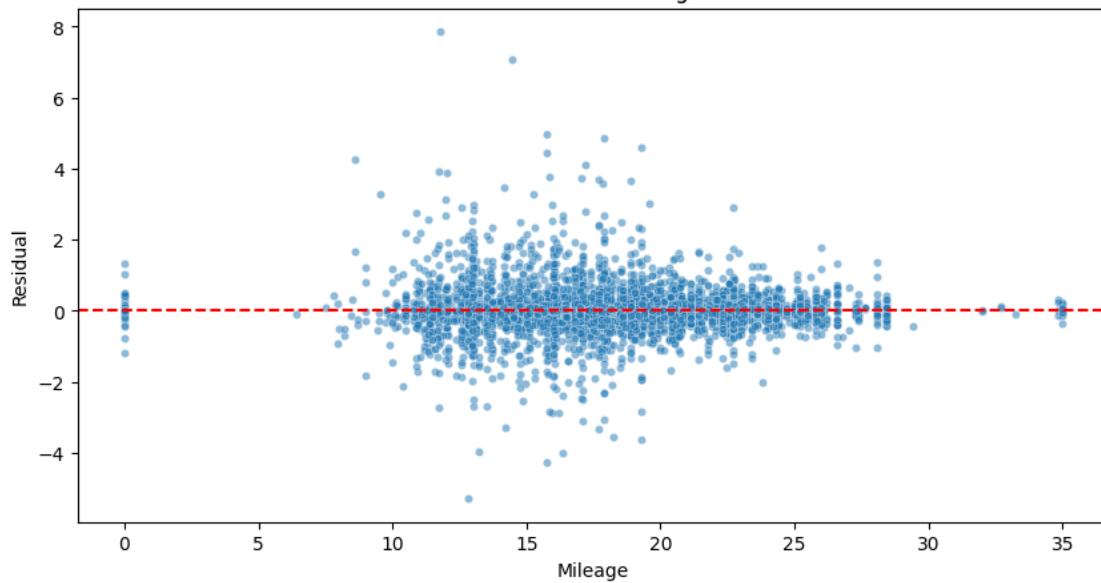




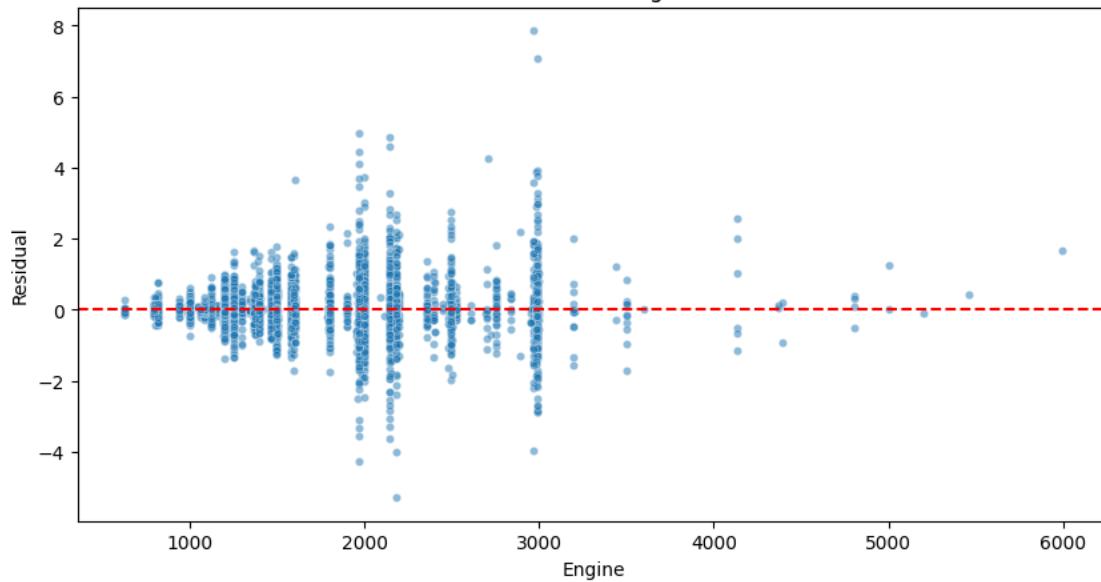


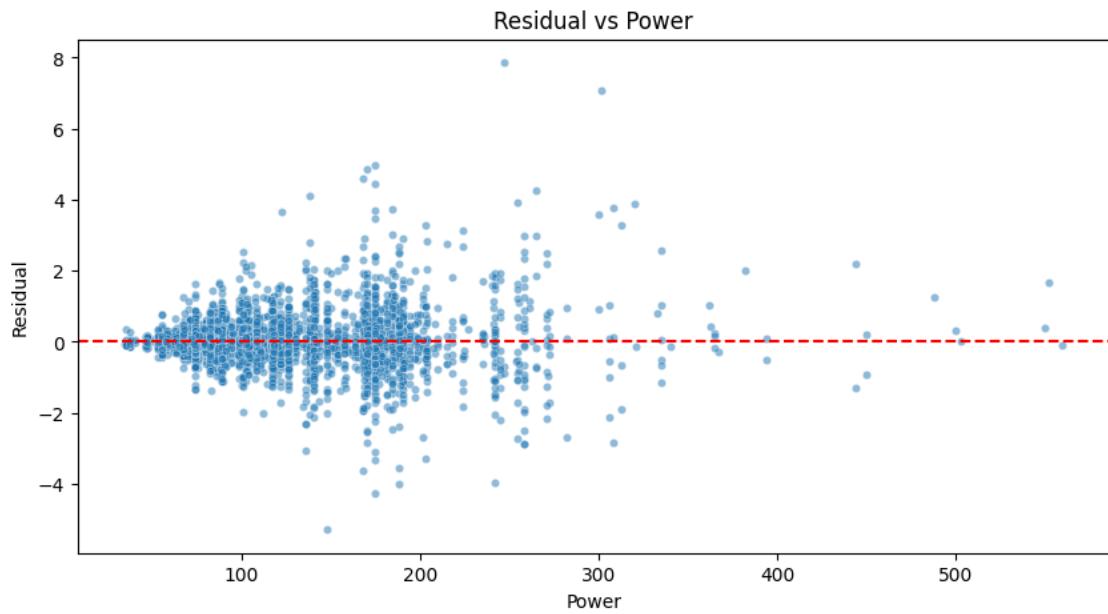


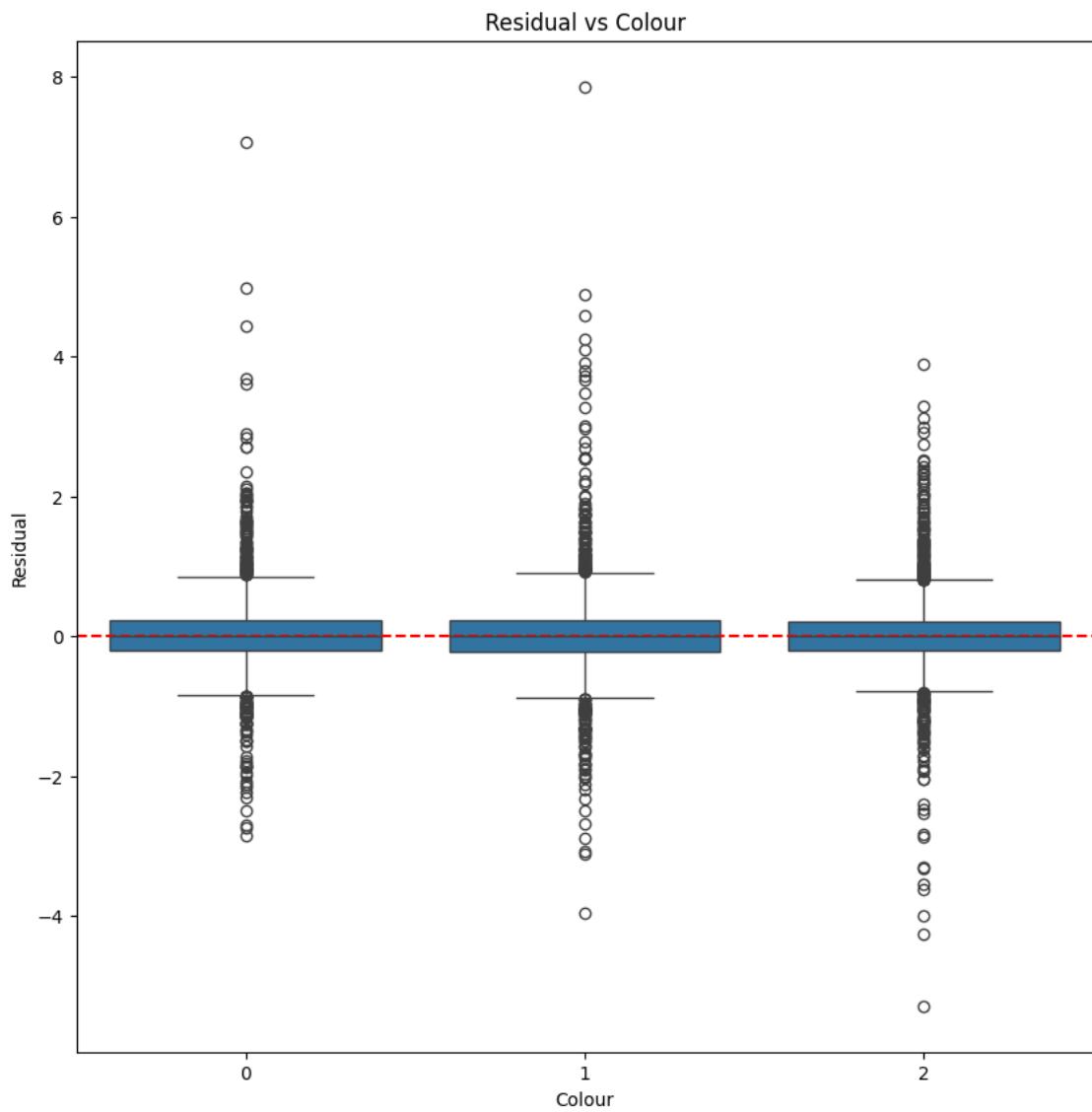
Residual vs Mileage

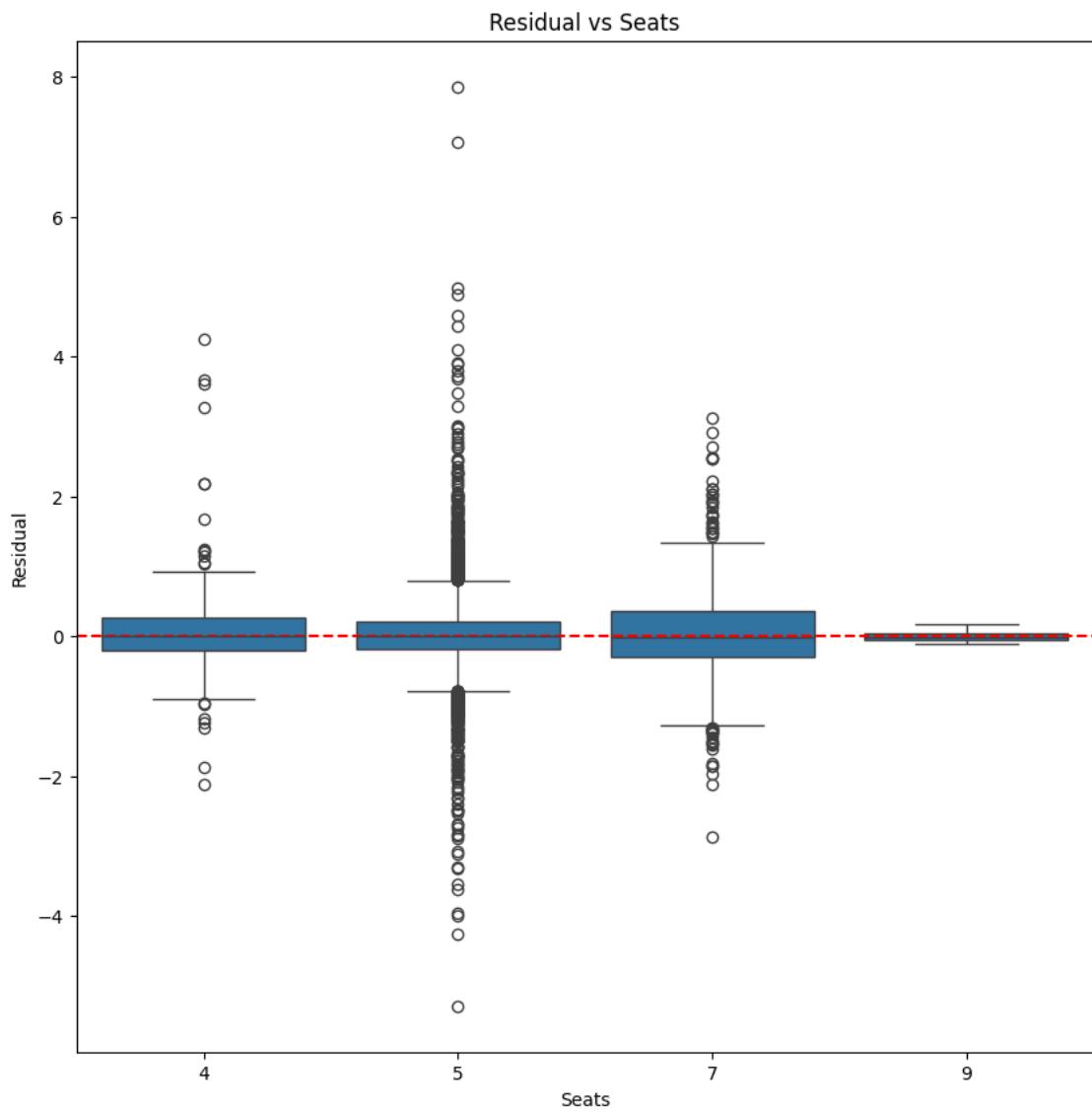


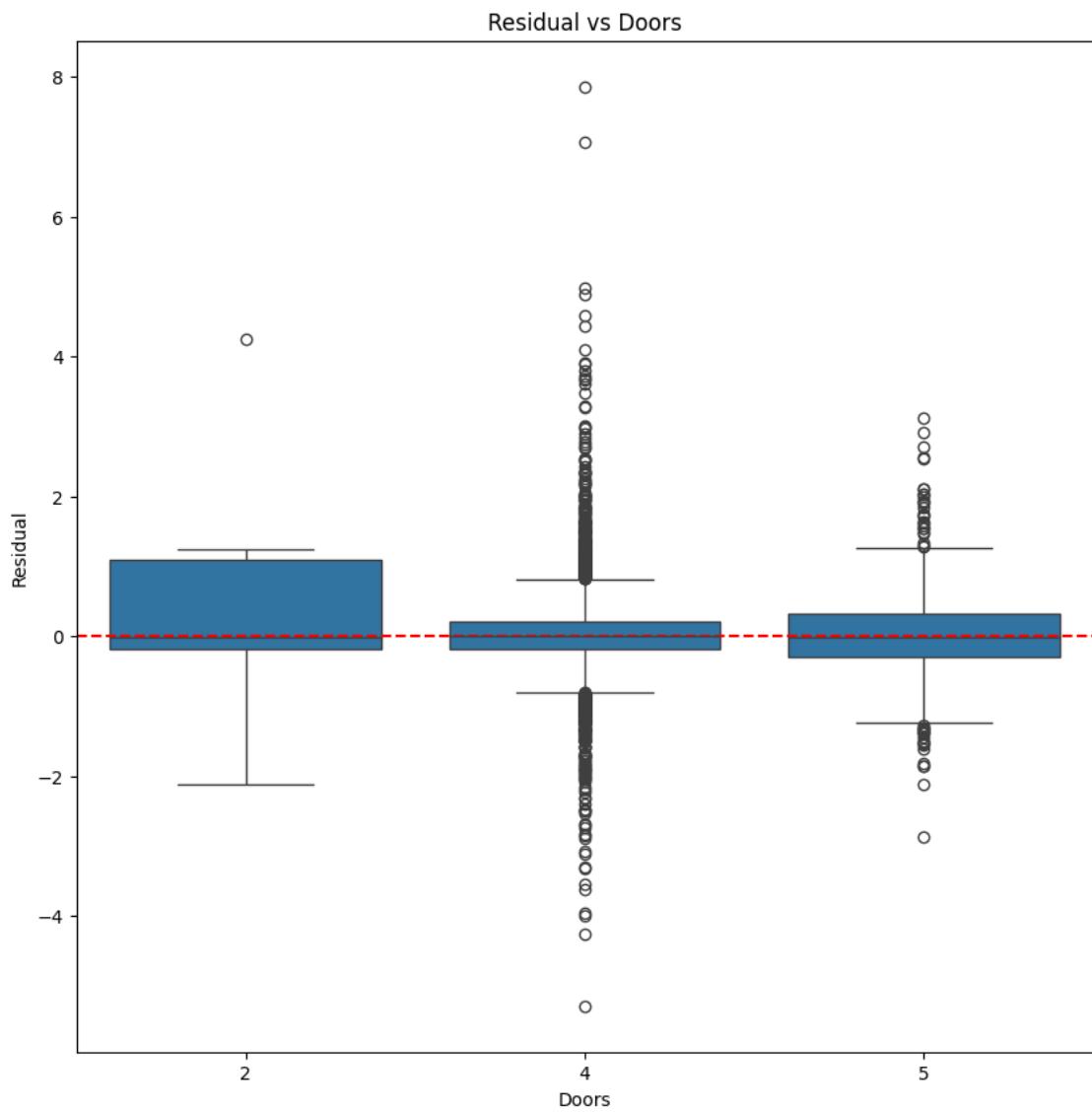
Residual vs Engine



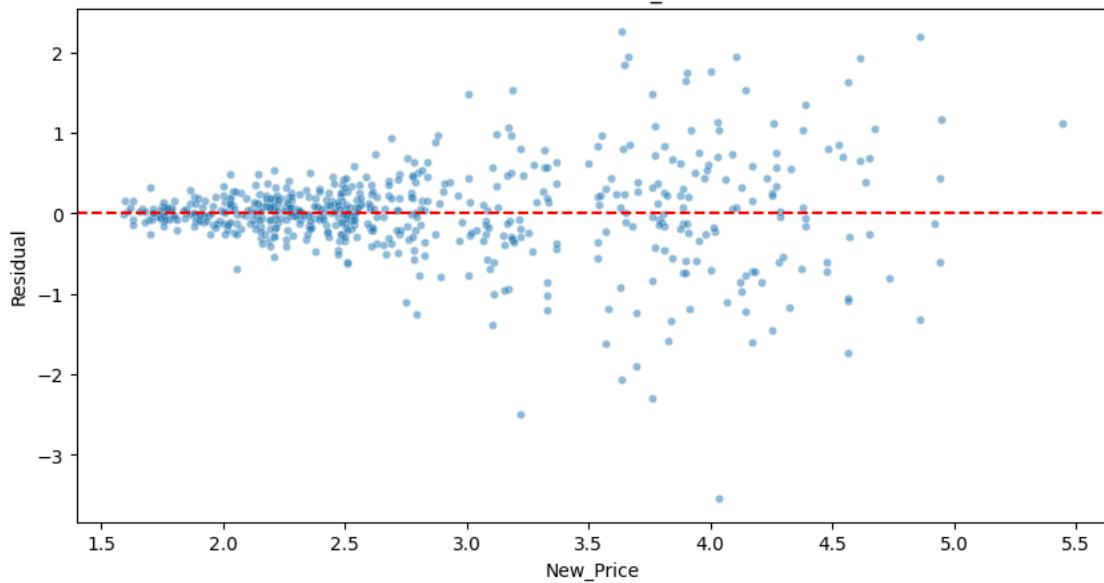




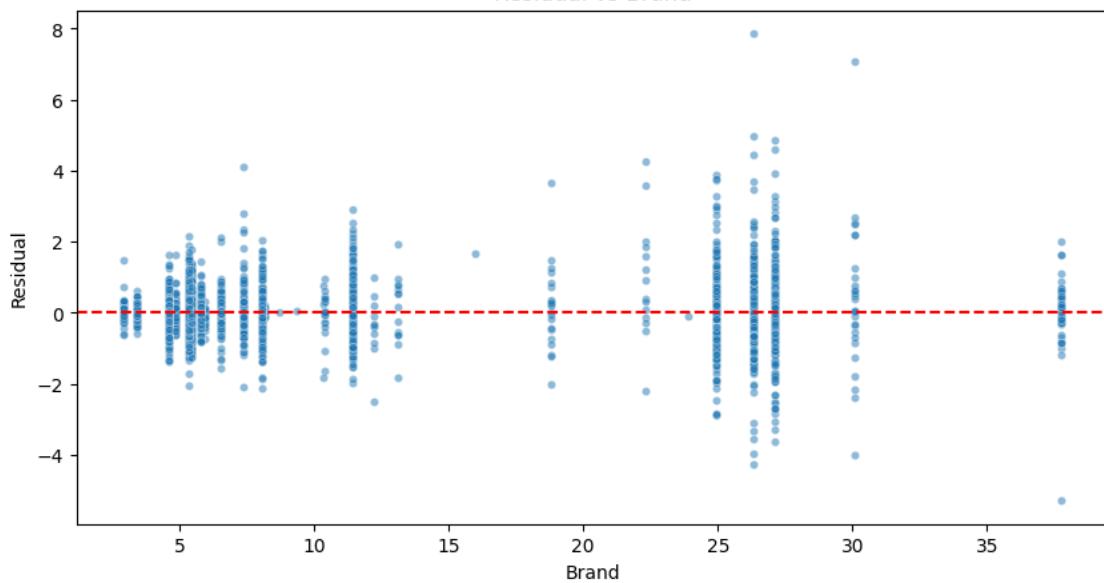


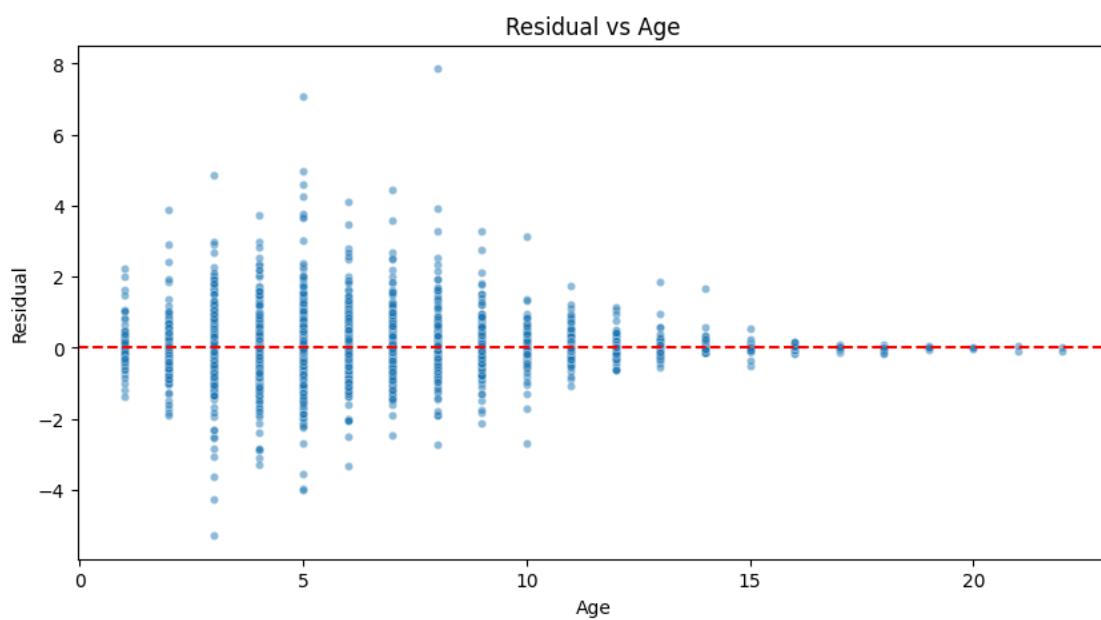
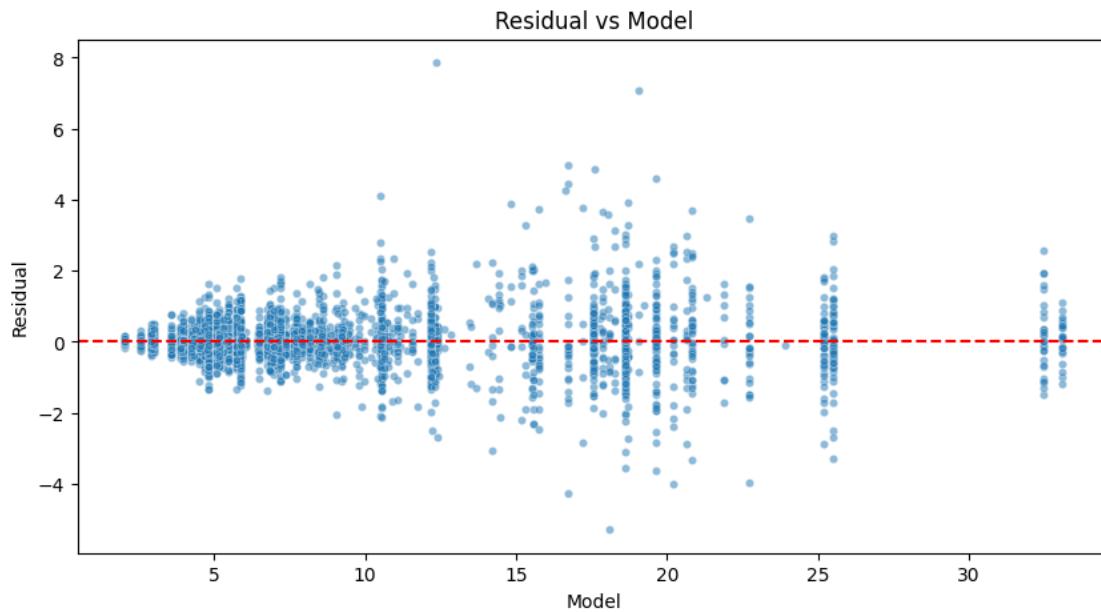


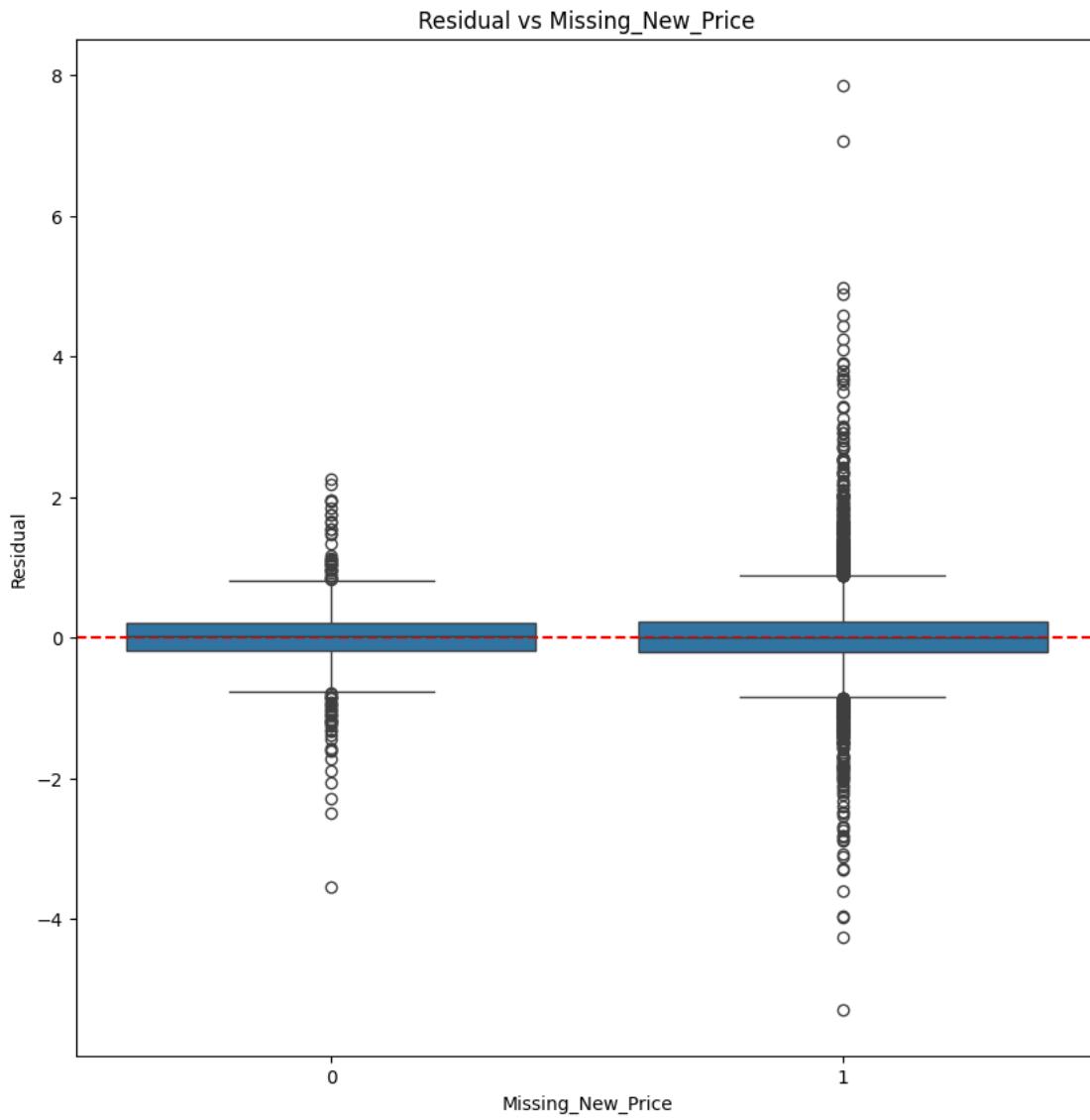
Residual vs New_Price



Residual vs Brand

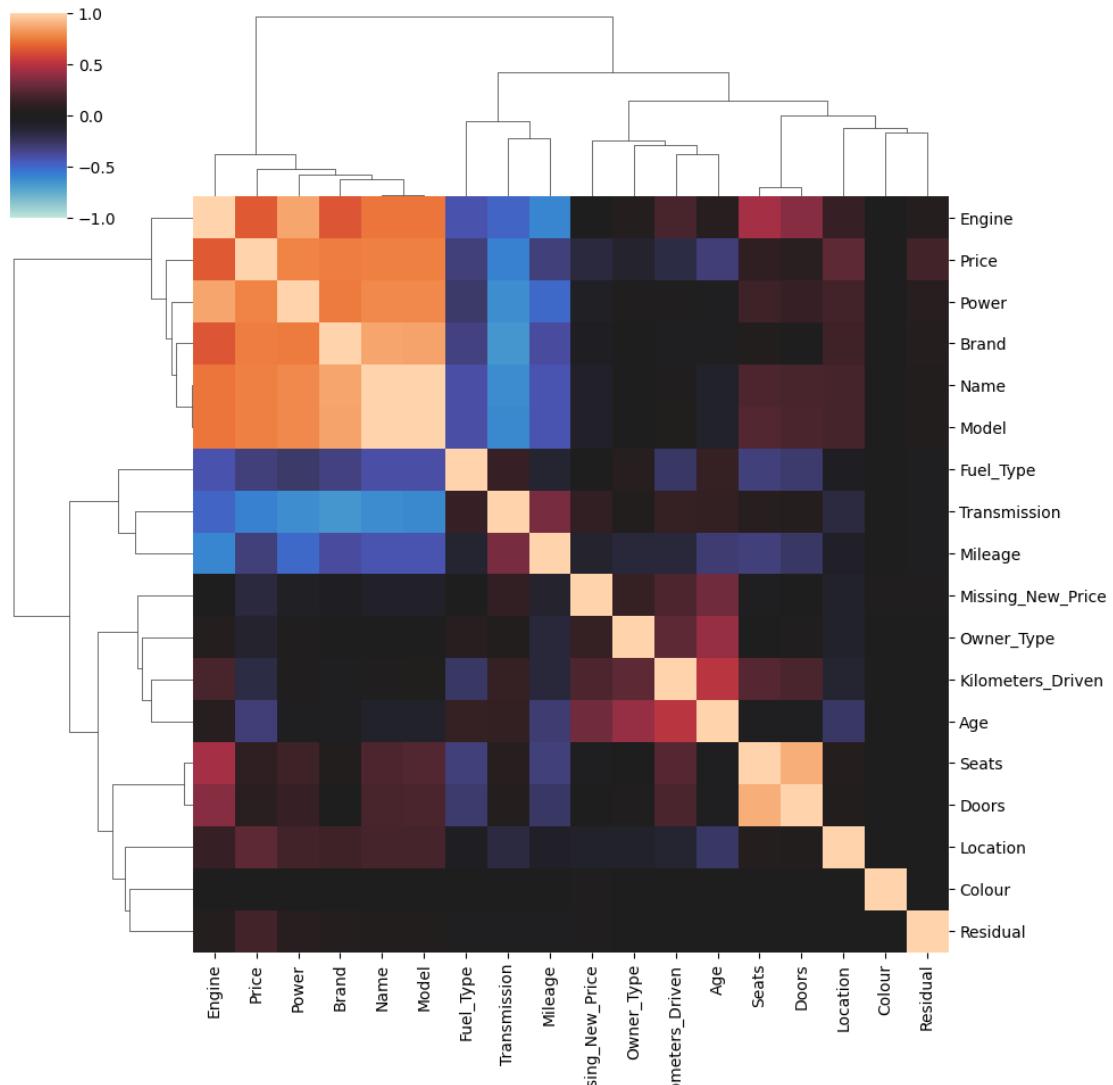






```
[88]: def plot_correlation(df, method="pearson", annot=True, **kwargs):
    sns.clustermap(
        df.corr(method, numeric_only=True),
        vmin=-1.0,
        vmax=1.0,
        cmap="icefire",
        method="complete",
        annot=annot,
        **kwargs,
    )
```

```
plot_correlation(df_residuals.drop(columns=["New_Price", "Prediction"]),
                 annot=False)
```



There is highly correlated block.

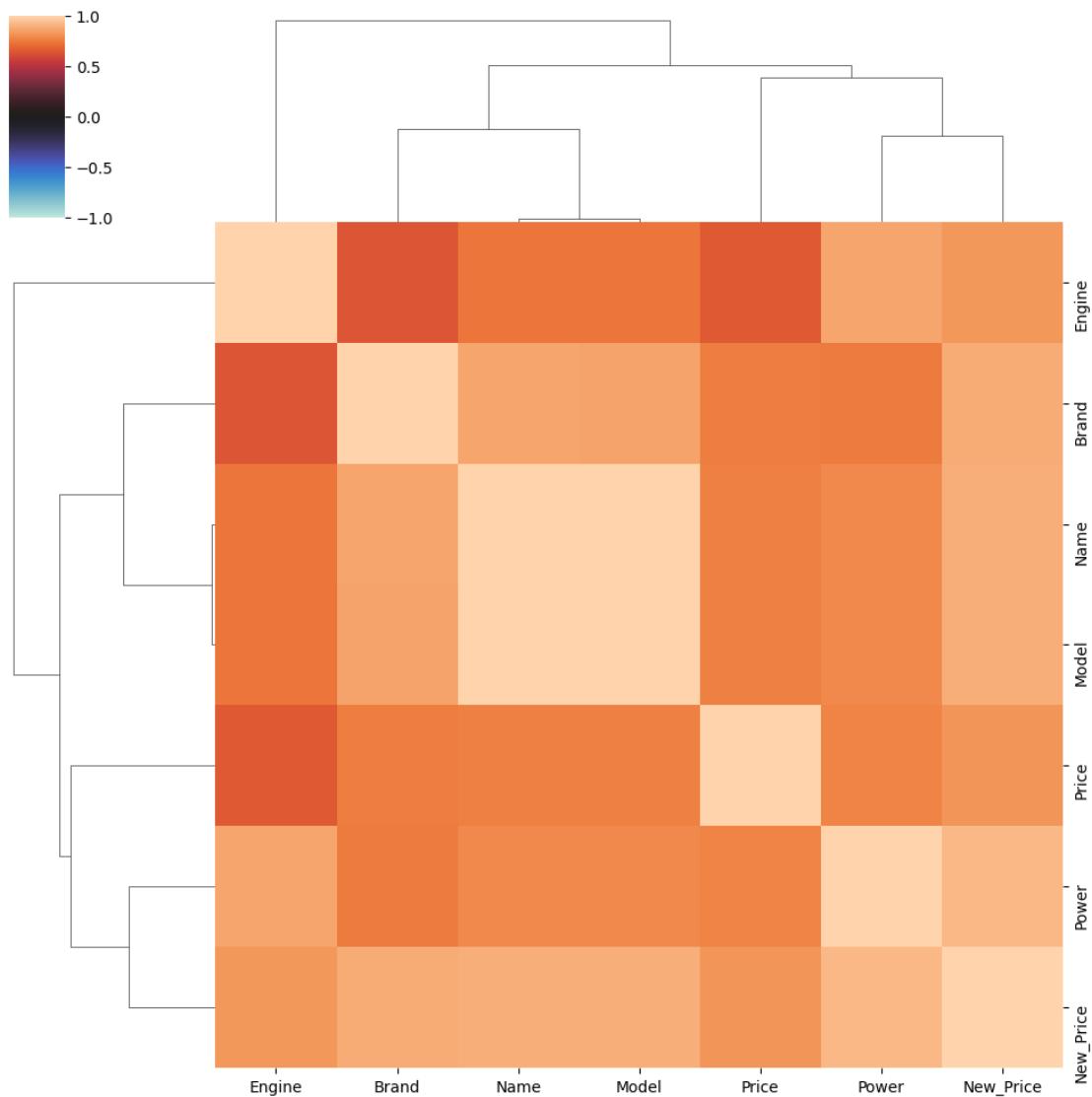
```
[89]: plot_correlation(
    df_residuals.drop(
        columns=[

            "Prediction",
            "Transmission",
            "Mileage",
            "Location",
            "Fuel_Type",
```

```

    "Seats",
    "Doors",
    "Missing_New_Price",
    "Owner_Type",
    "Kilometers_Driven",
    "Age",
    "Colour",
    "Residual",
]
),
annot=False,
)

```



Brand × Engine The residual correlation matrix strongly suggests correlation between Brand and Engine. This is plausible since the Engine given a Brand can give a strong signal on its tier within the Brand.

```
[90]: class BrandEngineInteraction(TransformerMixin, BaseEstimator):
    def __init__(self,
                 brand_col: str = "Brand",
                 engine_col: str = "Engine",
                 brand_engine_col: str = "Brand-Engine",
                 ):
        self.brand_col = brand_col
        self.engine_col = engine_col
        self.brand_engine_col = brand_engine_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> Union["BrandEngineInteraction", None]:
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.brand_engine_col] = X[self.brand_col] * X[self.engine_col]
        return X
```



```
[91]: brand_engine_interaction_exp = Experiment(
    ExperimentConfig(
        name="brand_engine_interaction",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("brand_engine_interaction", BrandEngineInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

brand_engine_interaction_exp.run(
    X_train, y_train, None, best_feature_by_feature_exp_result
);
```

```
[Experiment: brand_engine_interaction]
Cross-validating (5-folds)...
CV score: 0.1382 ± 0.0161
    +0.0004  -0.0012 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

It decreased the performance. This could be due to XGBoost already capturing such interaction implicitly, and the introduction of this interaction feature did not bring meaningfully more information.

Brand × Power Similarly, Brand and Power shows strong correlation and this is plausible with the same reason as why Brand and Engine interaction is meaningful.

```
[92]: class BrandPowerInteraction(TransformerMixin, BaseEstimator):
    def __init__(self,
                 brand_col: str = "Brand",
                 power_col: str = "Power",
                 brand_power_col: str = "Brand-Pwer",
                 ):
        self.brand_col = brand_col
        self.power_col = power_col
        self.brand_power_col = brand_power_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> Union["BrandPowerInteraction", None]:
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.brand_power_col] = X[self.brand_col] * X[self.power_col]
        return X
```

```
[93]: brand_power_interaction_exp = Experiment(
    ExperimentConfig(
        name="brand_power_interaction",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("brand_power_interaction", BrandPowerInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

brand_power_interaction_exp.run(
    X_train, y_train, None, best_feature_by_feature_exp_result
);
```

```
[Experiment: brand_power_interaction]
```

```
Cross-validating (5-folds)...
```

```

CV score: 0.1371 ± 0.0115
    -0.0008 -0.0058 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete

```

It improved the performance.

Age × New_Price Age and New_Price can represent depreciation. One of the most strongest model agnostic signal in predicting used-car price can be starting from New_Price and monotonically decreasing as it Ages.

```
[94]: class AgeNewPriceInteraction(BaseEstimator, TransformerMixin):
    def __init__(self,
                 age_col="Age",
                 new_price_col="New_Price",
                 out_col="Age-NewPrice",
                 ):
        self.age_col = age_col
        self.new_price_col = new_price_col
        self.out_col = out_col

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        X[self.out_col] = X[self.new_price_col] * X[self.age_col]
        return X
```

```
[95]: age_new_price_interaction_exp = Experiment(
    ExperimentConfig(
        name="age_new_price_interaction",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("age_new_price_interaction", AgeNewPriceInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

age_new_price_interaction_exp.run(
    X_train, y_train, None, best_feature_by_feature_exp_result
);
```

```
[Experiment: age_new_price_interaction]
Cross-validating (5-folds)...
CV score: 0.1368 ± 0.0159
    -0.0010 -0.0014 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

It slightly increased the performance and stability.

```
[96]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("brand_power_interaction", BrandPowerInteraction()),
                ("age_new_price_interaction", AgeNewPriceInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

current_best_exp.run(X_train, y_train, None, ↴
    ↪best_feature_by_feature_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1380 ± 0.0105
    +0.0002 -0.0068 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

I'll try more directly modeling depreciation.

The ratio of Price to New_Price could tell the normalized amount of depreciation. Assuming constant rate of price drop per year.

```
[97]: X_train_intersection = X_train.dropna(subset=["New_Price"])
y_train_intersection = y_train.loc[X_train_intersection.index]
current_year = 2020

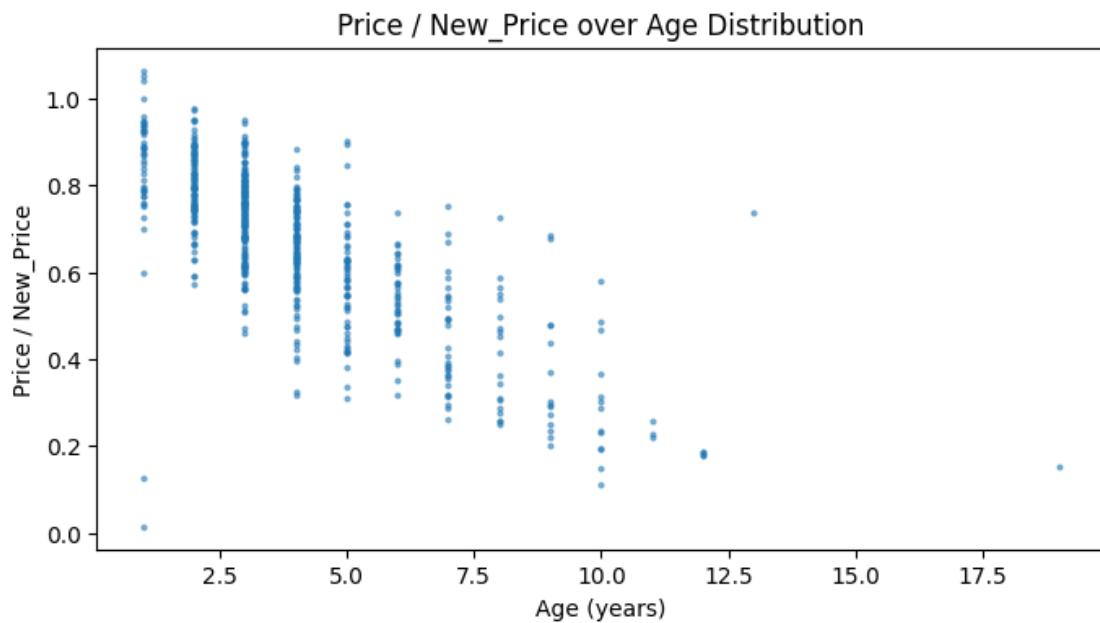
ages = current_year - X_train_intersection.Year
prices_over_new_price = y_train_intersection / X_train_intersection.New_Price
```

```

plt.figure(figsize=(8, 4))
plt.scatter(ages, prices_over_new_price, s=4, alpha=0.5)

plt.xlabel("Age (years)")
plt.ylabel("Price / New_Price")
plt.title("Price / New_Price over Age Distribution")
plt.show()

```



```

Model × Power
[98]: class ModelPowerInteraction(BaseEstimator, TransformerMixin):
    def __init__(
        self,
        model_col="Model",
        power_col="Power",
        model_power_col="Model-Power",
    ):
        self.model_col = model_col
        self.power_col = power_col
        self.model_power_col = model_power_col

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()

```

```
X[self.model_power_col] = X[self.model_col] * X[self.power_col]
return X
```

```
[99]: model_power_interaction_exp = Experiment(
    ExperimentConfig(
        name="model_power_interaction",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("model_power_interaction", ModelPowerInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

model_power_interaction_exp.run(
    X_train, y_train, None, best_feature_by_feature_exp_result
);
```

```
[Experiment: model_power_interaction]
Cross-validating (5-folds)...
CV score: 0.1360 ± 0.0141
    -0.0018 -0.0032 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

```
[100]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=Pipeline(
            [
                *best_single_feature_pipeline.steps[:-2],
                ("brand_power_interaction", BrandPowerInteraction()),
                ("model_power_interaction", ModelPowerInteraction()),
                *best_single_feature_pipeline.steps[-2:],
            ]
        ),
    )
)

current_best_exp.run(X_train, y_train, None,
    ↪best_feature_by_feature_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
```

```
CV score: 0.1372 ± 0.0116
      -0.0007 -0.0057 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

Ontology Approach

```
[101]: df_train.columns
```

```
[101]: Index(['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Colour',
       'Seats', 'Doors', 'New_Price', 'Price'],
       dtype='object')
```

Purely data-driven approach is failing. Attempting to build a strongly domain-knowledge based ontology of features.

```
[102]: used_car_price_ontology = {
    "Car_Spec": {
        "Power": ["Engine", "Power", "Fuel_Type", "Transmission"],
        "Form": ["Seats", "Doors", "Colour"],
        "Positioning": ["Brand", "Model", "Name", "New_Price"],
    },
    "Condition": {"Wear": ["Age", "Kilometers_Driven"], "Ownership": [
        "Owner_Type"]},
    "Market": ["Location"],
}
```

The Car_Spec will determine the car's BaseValue, the Condition will determine the car's Depreciation, and the Market will determine the MarketInfluence.

```
[103]: class BaseValueIndexTransformer(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        brand_col="Brand",
        model_col="Model",
        fuel_col="Fuel_Type",
        trans_col="Transmission",
        engine_col="Engine",
        power_col="Power",
        seats_col="Seats",
        doors_col="Doors",
        name_col="Name",
        new_price_col="New_Price",
        n_segments=3,
    ):
        self.brand_col = brand_col
```

```

    self.model_col = model_col
    self.fuel_col = fuel_col
    self.trans_col = trans_col
    self.engine_col = engine_col
    self.power_col = power_col
    self.seats_col = seats_col
    self.doors_col = doors_col
    self.name_col = name_col
    self.new_price_col = new_price_col
    self.n_segments = n_segments

def _spec_features(self, df: pd.DataFrame) -> pd.DataFrame:
    out = pd.DataFrame(index=df.index)

    out["LogEngine"] = np.log1p(df[self.engine_col])
    out["LogPower"] = np.log1p(df[self.power_col])
    out["PowerDensity"] = df[self.power_col] / df[self.engine_col]

    out[self.seats_col] = df[self.seats_col]
    out[self.doors_col] = df[self.doors_col]

    return out

def fit(
    self, X: pd.DataFrame, y: Optional[pd.Series] = None
) -> "BaseValueIndexTransformer":
    df = X.copy()

    mask = df[self.new_price_col].notna()
    df_known = df.loc[mask].copy()

    num_df = self._spec_features(df_known)

    cat_df = df_known[
        [self.brand_col, self.model_col, self.fuel_col, self.trans_col]
    ]

    self.ohe_ = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
    cat_arr = self.ohe_.fit_transform(cat_df)

    num_arr = num_df.fillna(0).values
    X_spec = np.hstack([cat_arr, num_arr])

    self.kmeans_ = KMeans(n_clusters=self.n_segments, random_state=42)
    segments = self.kmeans_.fit_predict(X_spec)
    df_known["Segment"] = segments

```

```

    self.num_features_ = num_df.columns.tolist()

    X_reg = np.hstack([X_spec, segments.reshape(-1, 1)])
    y_reg = df_known[self.new_price_col]

    self.reg_ = LinearRegression()
    self.reg_.fit(X_reg, y_reg)

    self.base_mean_ = y_reg.mean()

    return self

def transform(self, X: pd.DataFrame) -> pd.DataFrame:
    df = X.copy()

    num_df = self._spec_features(df)

    cat_df = df[[self.brand_col, self.model_col, self.fuel_col, self.
    ↪trans_col]]
    cat_arr = self.ohe_.transform(cat_df)

    num_arr = num_df.fillna(0).values

    X_spec = np.hstack([cat_arr, num_arr])

    segments = self.kmeans_.predict(X_spec)
    df["Segment"] = segments

    X_reg = np.hstack([X_spec, segments.reshape(-1, 1)])

    estimated_np = self.reg_.predict(X_reg)

    df["BaseValueIndex"] = estimated_np / self.base_mean_

    return df

```

```

[104]: class WearIndexTransformer(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        age_col="Age",
        km_col="Kilometers_Driven",
        engine_col="Engine",
        owner_col="Owner_Type",
    ):
        self.age_col = age_col
        self.km_col = km_col
        self.engine_col = engine_col

```

```

    self.owner_col = owner_col

def _build_wear_features(self, df: pd.DataFrame) -> pd.DataFrame:
    out = pd.DataFrame(index=df.index)

    age = df[self.age_col].fillna(0)
    km = df[self.km_col].fillna(0)

    engine = df[self.engine_col].fillna(df[self.engine_col].median())

    out["Age_scaled"] = age
    out["KM_per_Year"] = km / (age + 1)
    out["WearFactor"] = km / (engine + 1)

    return out

def fit(
    self, X: pd.DataFrame, y: Optional[pd.Series] = None
) -> "WearIndexTransformer":
    df = X.copy()

    wear_df = self._build_wear_features(df)

    owner_vals = df[self.owner_col].astype("category").cat.codes
    wear_df["OwnerEncoded"] = owner_vals

    self.scaler_ = StandardScaler()
    wear_scaled = self.scaler_.fit_transform(wear_df)

    self.reg_ = LinearRegression()
    self.reg_.fit(wear_scaled, np.arange(len(wear_scaled)))

    self.w_ = self.reg_.coef_

    return self

def transform(self, X: pd.DataFrame) -> pd.DataFrame:
    df = X.copy()

    wear_df = self._build_wear_features(df)
    owner_vals = df[self.owner_col].astype("category").cat.codes
    wear_df["OwnerEncoded"] = owner_vals

    wear_scaled = self.scaler_.transform(wear_df)

    wear_score = wear_scaled @ self.w_

```

```

        wear_index = (wear_score - wear_score.min()) / (
            wear_score.max() - wear_score.min()
        )

        df["WearIndex"] = wear_index

    return df

```

```

[105]: class MarketIndexTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, location_col="Location"):
        self.location_col = location_col

    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "MarketIndexTransformer":
        df = X.copy()
        df["Price"] = y

        loc_means = df.groupby(self.location_col)["Price"].mean()

        df["LocMean"] = df[self.location_col].map(loc_means)

        self.ohe_ = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
        loc_ohe = self.ohe_.fit_transform(df[[self.location_col]])

        self.reg_ = LinearRegression()
        self.reg_.fit(loc_ohe, df["LocMean"])

        self.global_mean_ = df["Price"].mean()

    return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        df = X.copy()

        loc_ohe = self.ohe_.transform(df[[self.location_col]])

        loc_score = self.reg_.predict(loc_ohe)

        df["MarketIndex"] = loc_score / self.global_mean_

    return df

```

```

[106]: class OntologyFeatureBuilder(BaseEstimator, TransformerMixin):
    def __init__(
        self,
        base_transformer,

```

```

        wear_transformer,
        market_transformer,
        add_interactions=True,
        add_composite=True,
    ):
        self.base_transformer = base_transformer
        self.wear_transformer = wear_transformer
        self.market_transformer = market_transformer
        self.add_interactions = add_interactions
        self.add_composite = add_composite

    def fit(self, X, y):
        self.base_transformer.fit(X)

        self.wear_transformer.fit(X)

        self.market_transformer.fit(X, y)

    return self

def transform(self, X):
    df = X.copy()

    df = self.base_transformer.transform(df)
    df = self.wear_transformer.transform(df)
    df = self.market_transformer.transform(df)

    base = df["BaseValueIndex"]
    wear = df["WearIndex"]
    market = df["MarketIndex"]

    if self.add_interactions:
        df["Base_x_Wear"] = base * wear
        df["Base_x_Market"] = base * market
        df["Wear_x_Market"] = wear * market

    if self.add_composite:
        df["CompositeValue"] = base * wear * market

    return df

```

```
[107]: ontology_transformer = OntologyFeatureBuilder(
    base_transformer=BaseValueIndexTransformer(),
    wear_transformer=WearIndexTransformer(),
    market_transformer=MarketIndexTransformer(),
    add_interactions=True,
    add_composite=True,
```

```
)
```

```
[108]: ontology_exp = Experiment(  
    ExperimentConfig(  
        name="ontology-based-engineering",  
        pipeline=Pipeline(  
            [  
                *best_single_feature_pipeline.steps[:-2],  
                ("ontology", ontology_transformer),  
                *best_single_feature_pipeline.steps[-2:],  
            ]  
        ),  
    ),  
)  
  
ontology_exp.run(X_train, y_train, None, best_feature_by_feature_exp_result);
```

```
[Experiment: ontology-based-engineering]  
Cross-validating (5-folds)...  
CV score: 0.1475 ± 0.0179  
    +0.0096  +0.0005 compared to combine-all-feature-by-feature-engineering  
(Negative is better)  
Training on full training set...  
Experiment complete
```

Final Feature Engineering Pipeline

```
[109]: def final_pipeline_builder(model_config=dict()) -> Pipeline:  
    return Pipeline(  
        [  
            ("extract_brand_model", BrandModelExtractor()),  
            (  
                "target_encode",  
                TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),  
            ),  
            ("transform", YearToAgeTransformer()),  
            ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),  
            ("group_infrequent_fuel_type", FuelTypeGrouper()),  
            ("mileage_clip_outliers", MileageClipper()),  
            ("imput_power", PowerImputer()),  
            ("bin_seats", SeatsBinner()),  
            ("transform_new_price", NewPriceTransformer()),  
            ("brand_power_interaction", BrandPowerInteraction()),  
            ("model_power_interaction", ModelPowerInteraction()),  
            ("category_encode", CategoricalEncoder()),  
            (  
        ]
```

```
        "model",
        TransformedTargetRegressor(
            regressor=XGBRegressor(**model_config),
            func=np.log1p,
            inverse_func=np.expm1,
        ),
    ),
],
)
```

```
[110]: current_best_exp = Experiment(
    ExperimentConfig(
        name="current-best",
        pipeline=final_pipeline_builder(),
    )
)

current_best_exp.run(X_train, y_train, None, ↴
    best_feature_by_feature_exp_result);
```

```
[Experiment: current-best]
Cross-validating (5-folds)...
CV score: 0.1372 ± 0.0116
    -0.0007  -0.0057 compared to combine-all-feature-by-feature-engineering
(Negative is better)
Training on full training set...
Experiment complete
```

A.4 Model Comparison Notebook (`models.ipynb`)

models

November 17, 2025

0.0.1 Model Comparison

```
[1]: %load_ext autoreload
%autoreload 2

[2]: from mldl_hw3.preprocessing import DataLoader
from mldl_hw3.feature_engineering import build_feature_engineering_pipeline
from mldl_hw3.experiment import Experiment, ExperimentConfig

from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor
import pandas as pd

[3]: df_train, df_test = DataLoader("../dataset").load()

X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.drop(columns=["Price"])


```

XGBoost

```
[4]: xgb_exp = Experiment(
    ExperimentConfig(
        name="xg-boost",
        pipeline=build_feature_engineering_pipeline(XGBRegressor())
    )
)

xgb_exp_result = xgb_exp.run(X_train, y_train, X_test)
```

```
[Experiment: xg-boost]
Cross-validating (5-folds)...
CV score: 0.1372 ± 0.0116
Training on full training set...
Creating submission on test set...
Submission created: artifacts/experiment-results/xg-boost.csv
Experiment complete
```

Random Forest

```
[5]: random_forest_exp = Experiment(  
    ExperimentConfig(  
        name="random-forest",  
        pipeline=build_feature_engineering_pipeline(RandomForestRegressor()),  
    )  
)  
  
random_forest_exp_result = random_forest_exp.run(X_train, y_train, X_test)
```

```
[Experiment: random-forest]  
Cross-validating (5-folds)...  
CV score: 0.1527 ± 0.0146  
Training on full training set...  
Creating submission on test set...  
Submission created: artifacts/experiment-results/random-forest.csv  
Experiment complete
```

CatBoost

```
[6]: cat_boost_exp = Experiment(  
    ExperimentConfig(  
        name="cat-boost",  
        pipeline=build_feature_engineering_pipeline(  
            CatBoostRegressor(  
                silent=True,  
                train_dir="../artifacts/catboost",  
                loss_function="MAPE",  
            )  
        ),  
    ),  
)  
  
cat_boost_exp_result = cat_boost_exp.run(X_train, y_train, X_test)
```

```
[Experiment: cat-boost]  
Cross-validating (5-folds)...  
CV score: 0.1396 ± 0.0164  
Training on full training set...  
Creating submission on test set...  
Submission created: artifacts/experiment-results/cat-boost.csv  
Experiment complete
```

```
[7]: pd.DataFrame(  
    {  
        "Model": ["XGBoost", "Random Forest", "CatBoost"],
```

```
"MAPE": [
    xgb_exp_result.cv_score,
    random_forest_exp_result.cv_score,
    cat_boost_exp_result.cv_score,
],
"MAPE std": [
    xgb_exp_result.cv_std,
    random_forest_exp_result.cv_std,
    cat_boost_exp_result.cv_std,
],
},
).set_index("Model").sort_values(by="MAPE")
```

[7]: MAPE MAPE std

Model	MAPE	MAPE std
XGBoost	0.137150	0.011620
CatBoost	0.139573	0.016407
Random Forest	0.152714	0.014628

A.5 Hyperparameter Tuning Notebook (hyperparameter-tuning.ipynb)

hyperparameter-tuning

November 17, 2025

0.0.1 Hyperparameter Tuning

```
[1]: %load_ext autoreload
%autoreload 2

[2]: from mldl_hw3.preprocessing import DataLoader
from mldl_hw3.feature_engineering import build_feature_engineering_pipeline
from mldl_hw3.experiment import Experiment, ExperimentConfig

from itertools import product

from xgboost import XGBRegressor
from tqdm import tqdm
from IPython.display import clear_output
import pandas as pd

[3]: df_train, df_test = DataLoader("../dataset").load()

X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.drop(columns=["Price"])
```

Grid Search

```
[4]: grid_config = {
    "max_depth": [4, 6],
    "min_child_weight": [1, 5],
    "gamma": [0.0, 0.1],
    "reg_lambda": [1, 2, 5],
    "reg_alpha": [0, 0.1],
    "subsample": [0.7, 0.9],
    "colsample_bytree": [0.7, 0.9],
    "learning_rate": [0.03, 0.1],
    "n_estimators": [800, 1500],
    "objective": ["reg:squarederror"],
    "tree_method": ["hist"],
    "random_state": [42],
}
```

```

grid = [
    dict(zip(grid_config.keys(), combination))
    for combination in product(*grid_config.values())
]

```

[5]:

```

grid_search_results = []

for i, params in tqdm(enumerate(grid), total=len(grid)):
    exp = Experiment(
        ExperimentConfig(
            name=f"xgb-params-grid-search-{i}",
            pipeline=build_feature_engineering_pipeline(XGBRegressor(**params)),
            extra={"xgb-params": params},
        )
    )

    exp_result = exp.run(X_train, y_train, skip_full_training=True)

    clear_output(wait=True)
    grid_search_results.append((params, exp_result))

```

100% | 768/768 [09:35<00:00, 1.33it/s]

[6]:

```

param_keys = grid_search_results[0][0].keys()

df_grid_search_results = pd.DataFrame(
{
    **{key: [d[key] for d, _ in grid_search_results] for key in param_keys},
    "MAPE": [exp_result.cv_score for _, exp_result in grid_search_results],
    "MAPE_std": [exp_result.cv_std for _, exp_result in
    grid_search_results],
}
).sort_values(by=["MAPE"])
df_grid_search_results

```

[6]:

	max_depth	min_child_weight	gamma	reg_lambda	reg_alpha	subsample	
46	4		1	0.0	2	0.0	0.9
1	4		1	0.0	1	0.0	0.7
47	4		1	0.0	2	0.0	0.9
34	4		1	0.0	2	0.0	0.7
35	4		1	0.0	2	0.0	0.7
..
312	4		5	0.1	1	0.1	0.9
120	4		1	0.1	1	0.1	0.9
570	6		1	0.1	5	0.1	0.9
376	4		5	0.1	5	0.1	0.9
360	4		5	0.1	5	0.0	0.9

```

  colsample_bytree  learning_rate  n_estimators      objective \
46            0.9          0.10        800  reg:squarederror
1             0.7          0.03       1500  reg:squarederror
47            0.9          0.10       1500  reg:squarederror
34            0.7          0.10        800  reg:squarederror
35            0.7          0.10       1500  reg:squarederror
..
312           0.7          0.03        800  reg:squarederror
120           0.7          0.03        800  reg:squarederror
570           0.7          0.10        800  reg:squarederror
376           0.7          0.03        800  reg:squarederror
360           0.7          0.03        800  reg:squarederror

  tree_method  random_state      MAPE   MAPE_std
46         hist           42  0.126625  0.015765
1          hist           42  0.127095  0.017024
47         hist           42  0.127118  0.016073
34         hist           42  0.127166  0.014453
35         hist           42  0.127531  0.014594
..
312        hist           42  0.149529  0.015915
120        hist           42  0.149531  0.016226
570        hist           42  0.149603  0.014402
376        hist           42  0.149861  0.015222
360        hist           42  0.150076  0.015829

```

[768 rows x 14 columns]

```

[7]: best_config = (
    df_grid_search_results.drop(columns=["MAPE", "MAPE_std"]).iloc[0].to_dict()
)
best_config

```

```

[7]: {'max_depth': 4,
      'min_child_weight': 1,
      'gamma': 0.0,
      'reg_lambda': 2,
      'reg_alpha': 0.0,
      'subsample': 0.9,
      'colsample_bytree': 0.9,
      'learning_rate': 0.1,
      'n_estimators': 800,
      'objective': 'reg:squarederror',
      'tree_method': 'hist',
      'random_state': 42}

```

```
[8]: xgb_pipeline = build_feature_engineering_pipeline(XGBRegressor(**best_config))
xgb_pipeline.fit(X_train, y_train)
test_predictions = xgb_pipeline.predict(X_test)
pd.DataFrame({"ID": X_test.index, "Price": test_predictions}).to_csv(
    "./artifacts/experiment-results/grid-search-tuning.csv", index=False
)
```

A.6 Preprocessing Code (*preprocessing.py*)

```

from pathlib import Path
import os
import zipfile

from .consts import fuel_densities, owner_type_order

import pandas as pd
import kaggle

class DataLoader:
    """
    DataLoader that downloads dataset from kaggle competition `gist-mldl-25f-hw3` and preprocesses.

    Args:
        dir (Path | str): directory of the dataset. The directory should contain `train.csv` and `test.csv`.
    """

    def __init__(self, dir: Path | str):
        if isinstance(dir, str):
            dir = Path(dir)
        if not dir.exists():
            dir.mkdir(parents=True)
        self.dir = dir
        self.competition = "gist-mldl-25f-hw3"

    def load(self, download: bool = False) -> tuple[pd.DataFrame, pd.DataFrame]:
        """
        Loads training and test dataset and preprocesses.

        Args:
            download (bool): Whether to download the dataset from kaggle. Default to False.

        Returns:
            (DataFrame, DataFrame): A tuple of the loaded and preprocessed training and test dataset.
        """
        if download:
            self.download()

        df_train = pd.read_csv(self.dir / "train.csv", index_col="ID")
        df_test = pd.read_csv(self.dir / "test.csv", index_col="ID")

        df_train[["_split"]] = "train"
        df_test[["_split"]] = "test"

        df = pd.concat([df_train, df_test])

        df = self.clean(df)
        df = self.encode(df)

        df_train = df.loc[df[["_split"]] == "train"].drop(columns=["_split"])
        df_test = df.loc[df[["_split"]] == "test"].drop(columns=["_split"])

        df_train = self.impute(df_train)

        return df_train, df_test

    def download(self) -> tuple[Path, Path]:
        """
        Downloads dataset from kaggle competition.

        Args:
            competition (str): Kaggle competition to download dataset from.
            dir (Path): Path to download the dataset at.

        Returns:
            (Path, Path): Tuple of training dataset and test dataset paths.
        """
        kaggle.api.authenticate()
        kaggle.api.competition_download_files(self.competition, self.dir)
        zip_file_path = Path(self.dir, self.competition).with_suffix(".zip")
        with zipfile.ZipFile(zip_file_path, "r") as zip_ref:
            zip_ref.extractall(self.dir)
        os.remove(zip_file_path)

        return (self.dir / "train.csv", self.dir / "test.csv")

    def clean(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Cleans the data to fix any errors or inconsistencies.
        - Replaces '\\N' missing value indicators with 'None'
        - Normalizes units ('Mileage': km/kg to kmpl, 'New_Price': Cr to Lakh)
        - Extracts numeric values from text ('Engine', 'Power')
        - Renames columns for consistency

        Args:
            df (DataFrame): The dataframe to be processed.

        Returns:
            DataFrame: The cleaned dataframe.
        """
        df = df.replace("\\N", None)

        df["Mileage"] = self._normalize_mileage(df)
        df["Engine"] = pd.to_numeric(
            df["Engine"].str.split(expand=True)[0], errors="coerce"
        )
        df["Power"] = pd.to_numeric(
            df["Power"].str.split(expand=True)[0], errors="coerce"
        )
        df["New_Price"] = self._normalize_new_price(df)

        df = df.rename(columns={"No. of Doors": "Doors"})

        return df

    def encode(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Encodes the statistical data types (numeric and categorical).
        - Sets appropriate data types for numeric columns
        - Sets categorical data types for categorical columns
        - Sets ordinal encoding for `Owner_Type`

        Args:
            df (DataFrame): The dataframe to be processed.

        Returns:
            DataFrame: The encoded dataframe.
        """
        df["Kilometers_Driven"] = pd.to_numeric(
            df["Kilometers_Driven"], errors="coerce"
        ).astype("Int64")
        df["Year"] = pd.to_numeric(df["Year"], errors="coerce").astype("Int64")

```

```

df["Engine"] = df["Engine"].astype("Int64")
df["Seats"] = pd.to_numeric(df["Seats"], errors="coerce").astype("Int64")
df["Doors"] = pd.to_numeric(df["Doors"], errors="coerce").astype("Int64")

df["Name"] = df["Name"].astype("category")
df["Location"] = df["Location"].astype("category")
df["Fuel_Type"] = df["Fuel_Type"].astype("category")
df["Transmission"] = df["Transmission"].astype("category")
df["Colour"] = df["Colour"].astype("category")

ordinal_owner_category = pd.CategoricalDtype(
    categories=owner_type_order, ordered=True
)
df["Owner_Type"] = df["Owner_Type"].astype(ordinal_owner_category)

return df

def impute(self, df: pd.DataFrame) -> pd.DataFrame:
    """
    Imputes any missing values.
    - Drops rows with missing features other than `Power` and `New_Price` (very few)
    - Other missing values kept as-is for further processing

    Args:
        df (DataFrame): The dataframe to be processed.

    Returns:
        DataFrame: The imputed dataframe.
    """
    return df.dropna(subset=list(set(df.columns) - {"Power", "New_Price"}))

def _normalize_mileage(self, df: pd.DataFrame) -> pd.Series:
    """
    Normalizes mileage by converting km/kg to kmpl based on fuel type density.

    Note: Even though this is not strictly in the realm of preprocessing, this is included here for simplicity. It is getting rid of inconsistency anyway.

    Args:
        df (DataFrame): The dataframe containing Mileage and Fuel_Type columns.
    Returns:
        Series: Normalized mileage values in kmpl.
    """
    mileage_split = df["Mileage"].str.split(expand=True)
    mileage_value = pd.to_numeric(mileage_split[0], errors="coerce")
    mileage_unit = mileage_split[1]

    conversion_factors = fuel_densities

    normalized_mileage = mileage_value.copy()

    for fuel_type, factor in conversion_factors.items():
        mask = (mileage_unit == "km/kg") & (df["Fuel_Type"] == fuel_type)
        normalized_mileage.loc[mask] = mileage_value.loc[mask] * factor

    return normalized_mileage

def _normalize_new_price(self, df: pd.DataFrame) -> pd.Series:
    """
    Normalizes New_Price to Lakh by converting Cr to Lakh (1 Cr = 100 Lakh).

    Args:
        df (DataFrame): The dataframe containing New_Price column.
    Returns:
        Series: Normalized price values in Lakh.
    """
    price_split = df["New_Price"].str.split(expand=True)
    price_value = pd.to_numeric(price_split[0], errors="coerce")
    price_unit = price_split[1]

    normalized_price = price_value.copy()
    cr_mask = price_unit == "Cr"
    normalized_price.loc[cr_mask] = price_value.loc[cr_mask] * 100

    return normalized_price

```

A.7 Experiment Code (**experiment.py**)

```

from pathlib import Path
from dataclasses import dataclass
from typing import Optional

import pandas as pd
from sklearn.model_selection import cross_validate
from sklearn.pipeline import Pipeline

@dataclass
class ExperimentConfig:
    name: str
    pipeline: Pipeline
    cv_folds: int = 5
    scoring: str = "neg_mean_absolute_percentage_error"
    extra: Optional[dict] = None

@dataclass
class ExperimentResult:
    name: str
    pipeline: Pipeline
    cv_score: float
    cv_std: float

class Experiment:
    def __init__(self,
                 config: ExperimentConfig,
                 output_dir: Path | str = "./artifacts/experiment-results",
                 ):
        self.config = config
        self.output_dir = (
            Path(output_dir) if isinstance(output_dir, str) else output_dir
        )

    def run(
            self,
            X_train: pd.DataFrame,
            y_train: pd.Series,
            X_test: Optional[pd.DataFrame] = None,
            baseline_exp_result: Optional[ExperimentResult] = None,
            skip_full_training: bool = False,
    ) -> ExperimentResult:
        print(f"[Experiment: {self.config.name}]")
        print(f"Cross-validating ({self.config.cv_folds}-folds)...")

        scores = cross_validate(
            self.config.pipeline,
            X_train,
            y_train,
            cv=self.config.cv_folds,
            scoring=self.config.scoring,
            n_jobs=-1,
        )

        cv_score = -scores["test_score"].mean().item()
        cv_std = scores["test_score"].std().item()

        print(f"CV score: {cv_score:.4f} ± {cv_std:.4f}")
        if baseline_exp_result is not None:
            delta_cv_score = cv_score - baseline_exp_result.cv_score
            delta_cv_std = cv_std - baseline_exp_result.cv_std
            print(
                f"          {delta_cv_score:+.4f} {delta_cv_std:+.4f} compared to {baseline_exp_result.name} (Negative is better)"
            )

        if not skip_full_training:
            print("Training on full training set...")
            self.config.pipeline.fit(X_train, y_train)

        if X_test is not None:
            print("Creating submission on test set...")
            submission_path = self.create_submission(X_test, self.output_dir)
            print(f"Submission created: {submission_path}")

        print("Experiment complete\n")

        return ExperimentResult(
            self.config.name, self.config.pipeline, cv_score, cv_std
        )

    def create_submission(self, X_test, dir: str | Path) -> Path:
        dir = Path(dir) if isinstance(dir, str) else dir
        if not dir.exists():
            dir.mkdir(parents=True)
        path = dir / f"{self.config.name}.csv"

        predictions = self.config.pipeline.predict(X_test)
        submission = pd.DataFrame({"ID": X_test.index, "Price": predictions})
        submission.to_csv(path, index=False)

        return path

```

A.8 Feature Engineering Code (*feature_engineering.py*)

```

from .consts import Brand
from typing import Optional

import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import TransformedTargetRegressor as TransformedTargetRegressor_
from category_encoders import TargetEncoder
from xgboost import XGBRegressor

class CategoricalEncoder(TransformerMixin, BaseEstimator):
    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "CategoricalEncoder":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        for feature in X.select_dtypes(["category"]):
            X[feature] = X[feature].cat.codes

        return X

class BrandModelExtractor(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        name_col: str = "Name",
        brand_col: str = "Brand",
        model_col: str = "Model",
        brand_enum=Brand,
    ):
        self.name_col = name_col
        self.brand_col = brand_col
        self.model_col = model_col
        self.brand_enum = brand_enum

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "BrandModelExtractor":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        names = X[self.name_col].str.title()
        brands = pd.Series([None] * len(names), index=names.index, dtype=object)
        models = pd.Series([None] * len(names), index=names.index, dtype=object)

        for brand in Brand:
            condition = names.str.startswith(brand.value, na=False) & brands.isna()

            if condition.any():
                brands.loc[condition] = brand.name
                matched_names = names.loc[condition]
                residuals = matched_names.str[len(brand.value) :].str.strip()
                models.loc[condition] = residuals.where(residuals != "", None)

        brands = brands.rename("Brand").astype("category")
        models = models.rename("Model").astype("category")

        X[self.brand_col] = brands
        X[self.model_col] = models

        return X

class YearToAgeTransformer(TransformerMixin, BaseEstimator):
    def __init__(
        self, year_col: str = "Year", age_col: str = "Age", current_year: int = 2020
    ):
        self.year_col = year_col
        self.age_col = age_col
        self.current_year = current_year

    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "YearToAgeTransformer":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        X[self.age_col] = self.current_year - X[self.year_col]
        X.drop(columns=[self.year_col], inplace=True)

        return X

class KilometersDrivenClipper(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        kilometers_driven_col: str = "Kilometers_Driven",
        clipping_quantile: float = 0.995,
    ):
        self.kilometers_driven_col = kilometers_driven_col
        self.clipping_quantile = clipping_quantile

    def fit(
        self, X: pd.DataFrame, y: Optional[pd.Series] = None
    ) -> "KilometersDrivenClipper":
        return self

    def transform(self, X: pd.DataFrame):
        X = X.copy()

        X[self.kilometers_driven_col] = X[self.kilometers_driven_col].clip(
            upper=int(X[self.kilometers_driven_col].quantile(self.clipping_quantile))
        )

        return X

class FuelTypeGrouper(TransformerMixin, BaseEstimator):
    def __init__(
        self,
        fuel_type_col: str = "Fuel_Type",
        target_fuel_types: list[str] = ["CNG", "LPG", "Electric"],
    ):
        self.fuel_type_col = fuel_type_col
        self.target_fuel_types = target_fuel_types

```

```

def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "FuelTypeGrouper":
    return self

def transform(self, X: pd.DataFrame) -> pd.DataFrame:
    X = X.copy()

    X[self.fuel_type_col] = (
        X[self.fuel_type_col]
        .astype("object")
        .replace(dict((fuel_type, "Other") for fuel_type in self.target_fuel_types))
        .astype("category")
    )

    return X

class MileageClipper(TransformerMixin, BaseEstimator):
    def __init__(self,
                 mileage_col: str = "Mileage",
                 clipping_quantile: float = 0.995,
                 ):
        self.mileage_col = mileage_col
        self.clipping_quantile = clipping_quantile

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series] = None) -> "MileageClipper":
        return self

    def transform(self, X: pd.DataFrame):
        X = X.copy()

        X[self.mileage_col] = X[self.mileage_col].clip(
            upper=int(X[self.mileage_col].quantile(self.clipping_quantile)))
        )

    return X

class PowerImputer(TransformerMixin, BaseEstimator):
    def __init__(self,
                 engine_col: str = "Engine", power_col="Power", clip_negative: bool = True
                 ):
        self.engine_col = engine_col
        self.power_col = power_col
        self.clip_negative = clip_negative

    def fit(self, X: pd.DataFrame, y=Optional[pd.Series]) -> "PowerImputer":
        df = X[[self.engine_col, self.power_col]].dropna()
        engines = df[self.engine_col].astype(float)
        powers = df[self.power_col].astype(float)

        # Linear regression
        cov = ((engines - engines.mean()) * (powers - powers.mean())).sum()
        var = ((engines - engines.mean()) ** 2).sum()
        self.slope = cov / var
        self.intercept = powers.mean() - self.slope * engines.mean()

        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        mask = X[self.power_col].isna()
        has_engine = X[self.engine_col].notna() & mask
        X.loc[has_engine, self.power_col] = (
            self.slope * X.loc[has_engine, self.engine_col] + self.intercept
        )

        if self.clip_negative:
            X[self.power_col] = X[self.power_col].clip(lower=0)

        return X

class SeatsBinner(TransformerMixin, BaseEstimator):
    def __init__(self, seats_col: str = "Seats"):
        self.seats_col = seats_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "SeatsBinner":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        seats = X[self.seats_col]

        X[self.seats_col] = seats.replace(
            {
                2: 4,
                4: 4,
                # Small
                5: 5,
                # Standard
                6: 7,
                7: 7,
                8: 7,
                # Large
                9: 9,
                10: 9,
                # Van
            }
        )

        return X

class NewPriceTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, new_price_col: str = "New_Price", func=np.log1p):
        self.new_price_col = new_price_col
        self.missing_col = "Missing_" + new_price_col
        self.func = func

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "NewPriceTransformer":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()

        X[self.missing_col] = X[self.new_price_col].isna().astype(int)
        X[self.new_price_col] = X[self.new_price_col].apply(self.func)

        return X

```

```

class BrandPowerInteraction(TransformerMixin, BaseEstimator):
    def __init__(self,
                 brand_col: str = "Brand",
                 power_col: str = "Power",
                 brand_power_col: str = "Brand-Pwer",
                 ):
        self.brand_col = brand_col
        self.power_col = power_col
        self.brand_power_col = brand_power_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "BrandPowerInteraction":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.brand_power_col] = X[self.brand_col] * X[self.power_col]
        return X

class ModelPowerInteraction(BaseEstimator, TransformerMixin):
    def __init__(self,
                 model_col="Model",
                 power_col="Power",
                 model_power_col="Model-Power",
                 ):
        self.model_col = model_col
        self.power_col = power_col
        self.model_power_col = model_power_col

    def fit(self, X: pd.DataFrame, y: Optional[pd.Series]) -> "ModelPowerInteraction":
        return self

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.model_power_col] = X[self.model_col] * X[self.power_col]
        return X

class TransformedPriceRegressor(TransformedTargetRegressor_):
    def __init__(self,
                 regressor,
                 func=np.log1p,
                 inverse_func=np.expm1,
                 ):
        super().__init__(regressor=regressor, func=func, inverse_func=inverse_func)

def build_feature_engineering_pipeline(regressor) -> Pipeline:
    return Pipeline(
        [
            ("extract_brand_model", BrandModelExtractor()),
            (
                "target_encode",
                TargetEncoder(cols=["Brand", "Model", "Name", "Location"]),
            ),
            ("transform", YearToAgeTransformer()),
            ("Kilometers_Driven_clip_outliers", KilometersDrivenClipper()),
            ("group_infrequent_fuel_type", FuelTypeGrouper()),
            ("mileage_clip_outliers", MileageClipper()),
            ("imput_power", PowerImputer()),
            ("bin_seats", SeatsBinner()),
            ("transform_new_price", NewPriceTransformer()),
            ("brand_power_interaction", BrandPowerInteraction()),
            ("model_power_interaction", ModelPowerInteraction()),
            ("category_encode", CategoricalEncoder()),
            ("model", TransformedPriceRegressor(regressor)),
        ]
    )
)

```

A.9 Constants Code (*consts.py*)

```
from enum import Enum

class Brand(Enum):
    """
    Enum of recognized brands. All the values are in title-case. Transform to title-case before comparing.
    """

    Audi = "Audi"
    Bentley = "Bentley"
    BMW = "Bmw"
    Chevrolet = "Chevrolet"
    Datsun = "Datsun"
    Fiat = "Fiat"
    Force = "Force"
    Ford = "Ford"
    Honda = "Honda"
    Hyundai = "Hyundai"
    Isuzu = "Isuzu"
    Jaguar = "Jaguar"
    Jeep = "Jeep"
    Lamborghini = "Lamborghini"
    Land_Rover = "Land Rover"
    Mahindra = "Mahindra"
    Maruti = "Maruti"
    Mercedes_Benz = "Mercedes-Benz"
    Mini = "Mini"
    Mitsubishi = "Mitsubishi"
    Nissan = "Nissan"
    Porsche = "Porsche"
    Renault = "Renault"
    Skoda = "Skoda"
    Smart = "Smart"
    Tata = "Tata"
    Toyota = "Toyota"
    Volkswagen = "Volkswagen"
    Volvo = "Volvo"

fuel_densities = {"CNG": 1.33, "Diesel": 1.20, "LPG": 1.85, "Petrol": 1.35}
owner_type_order = ["First", "Second", "Third", "Fourth & Above"]
```