# Car Price Prediction - Regression

Objectives

The primary goal of this project is to develop a robust regression model to predict used car prices for a reseller based on various listed features and specifications. In addition to predicting prices, the project focuses on identifying feature importance and mitigating overfitting through the application of regularisation techniques.

There can be several business objectives for this, such as:

* **Price Prediction**\*\*: Model car prices based on features like mileage, fuel type, and performance.
* **Market Analysis**\*\*: Explore trends and preferences in the used car market, by type, region, or other metrics.
* **Feature Importance**\*\*: Identify the most important factors influencing car prices (e.g., fuel type, mileage, age).

Data Dictionary

|  |  |
| --- | --- |
| Variable | Description |
| make\_model | The brand and model of the vehicle (e.g., 'Audi A1'). |
| body\_type | The body style of the vehicle, such as Sedan, Compact, or Station Wagon. |
| price | The listed price of the car in currency. |
| vat | Indicates the VAT status for the vehicle's price (e.g., VAT deductible, Price negotiable). |
| km | The total mileage (in kilometers) of the vehicle, indicating its usage. |
| Type | Condition of the vehicle, whether it's 'Used' or 'New'. |
| Fuel | Type of fuel the vehicle uses, such as 'Diesel', 'Benzine', etc. |
| Gears | The number of gears in the vehicle's transmission. |
| Comfort\_Convenience | Comfort and convenience features, such as 'Air conditioning', 'Leather steering wheel', 'Cruise control', and more. |
| Entertainment\_Media | Media features available in the vehicle, including 'Bluetooth', 'MP3', 'Radio', etc. |
| Extras | Additional features like 'Alloy wheels', 'Sport suspension', etc. |
| Safety\_Security | Safety features like 'ABS', 'Airbags', 'Electronic stability control', 'Isofix', etc. |
| age | Age of the car (calculated based on the model year). |
| Previous\_Owners | The number of previous owners the car has had. |
| hp\_kW | Engine power in kilowatts (kW), indicating the performance capacity of the engine. |
| Inspection\_new | Indicates whether the car has recently undergone an inspection (1 for yes, 0 for no). |
| Paint\_Type | The type of paint on the car, such as 'Metallic', 'Matte', etc. |
| Upholstery\_type | The material used for the interior upholstery, such as 'Cloth', 'Leather', etc. |
| Gearing\_Type | The type of transmission the car uses, either 'Automatic' or 'Manual'. |
| Displacement\_cc | The engine displacement in cubic centimeters (cc), indicating the size of the engine. |
| Weight\_kg | The total weight of the vehicle in kilograms. |
| Drive\_chain | The type of drivetrain, indicating whether it's 'Front' or 'Rear' wheel drive. |
| cons\_comb | The combined fuel consumption in liters per 100 kilometers. |

## Data Loading

* Define local and AWS environment configuration variables for base directory, raw data, processed data, models; for easy porting from local to AWS environment
* Import necessary libraries from pandas, numpy, mapplotlin, seaborn, sklearn, statsmodels, IPython, collections and warnings

### Load the Dataset

* df\_raw 🡨 Read the Car\_Price.csv file
* Assess the dataset information, columns, data types, row counts

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 15915 entries, 0 to 15914

Data columns (total 23 columns)

# Analysis and Feature Engineering

## Preliminary Analysis and Frequency Distributions

### Handle Missing Values

#### Standardize & Classify Columns, Define Imputation Strategy

* Standardize column names by converting to lowercase and replacing spaces with underscores
* Categorize the columns into
  + Category columns
  + Numeric columns
    - Numeric columns with imputation with mean
    - Numeric columns with imputation with median
    - Numeric columns with imputation with mode
  + Target column
  + Feature columns
* Use below Imputation Strategy
  + Categorical Columns Imputation Strategy - Use mode for all
  + Numeric Columns Imputation Strategy
    - If target column has missing values, better to drop the rows as imputing target variable can lead to bias
    - For other columns, follow below strategy:

|  |  |  |
| --- | --- | --- |
| Categorical Features | Suggested Imputation | Rationale |
| 'make\_model', 'body\_type', 'vat', 'type', 'fuel', 'comfort\_convenience', 'entertainment\_media', 'extras', 'safety\_security', 'paint\_type', 'upholstery\_type', 'gearing\_type', 'drive\_chain' | Mode | Discrete few values |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Numeric Features | Suggested Imputation | Rationale |
| km | Median | Mileage is usually skewed, median more robust than mean |
| Gears | Mode | Discrete small integers, categorical-like |
| age | Median | Skewed distribution possible, median more robust |
| Inspection\_new | Mode | Binary categorical (0/1) |
| Previous\_Owners | Mode | Small integers, categorical-like |
| hp\_kW | Mean or Median | Continuous; use mean if symmetric, median if skewed |
| Displacement\_cc | Mean | Continuous, usually symmetric within ranges |
| Weight\_kg | Mean | Continuous, symmetric, less outlier-prone |
| cons\_comb | Mean | Fuel consumption continuous, usually close to normal distribution |

|  |  |  |
| --- | --- | --- |
| Target | Suggested Imputation | Rationale |
| price | Median | Target variable; drop if only few missing records, but if you need imputation and it is a skewed distribution, median is safer than mean |

#### Fix Missing Values – Drop vs Impute

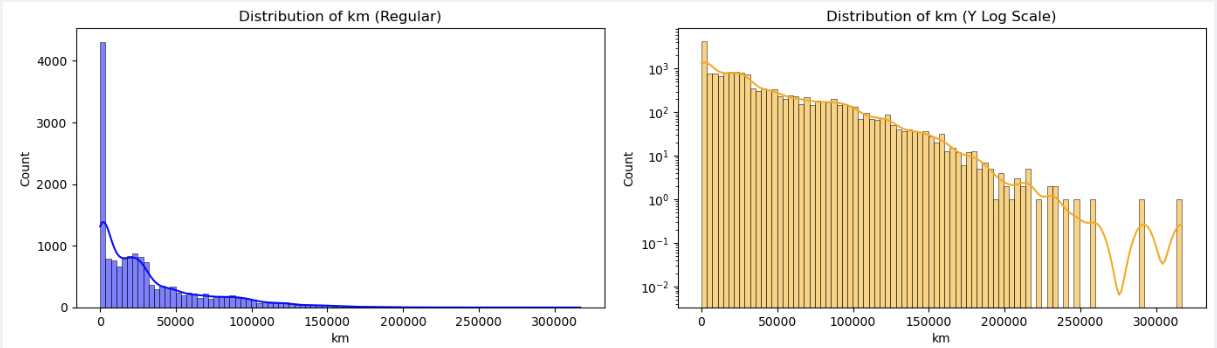
* Create a dataset with list of columns and their missing value proportions
* If target column has missing values, better to drop the rows as imputing target variable can lead to bias
* For all other columns, analyze missing value proportions and identify the action with below approach:
  + Missing % = 0%, no action needed
  + Missing % between 0-5%, drop the column
  + Missing % between 5-30%, impute using approach designed in 2.1.1
  + Missing % > 30%, Check data source, likely drop column

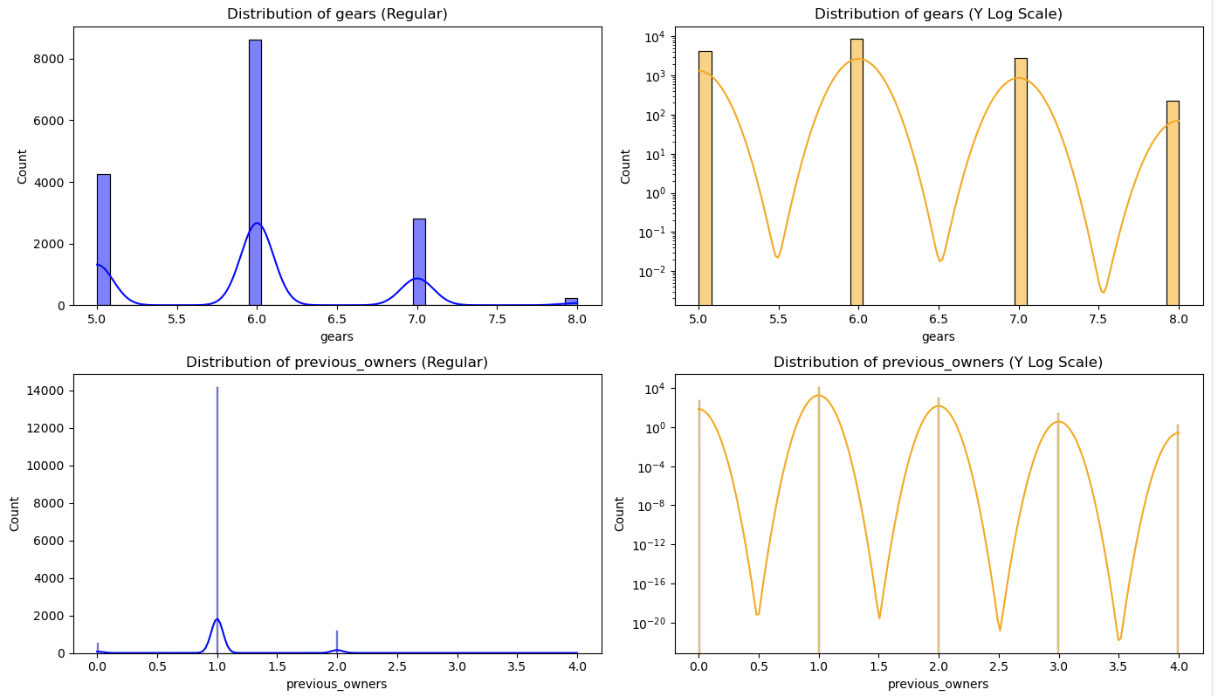
**Results:**

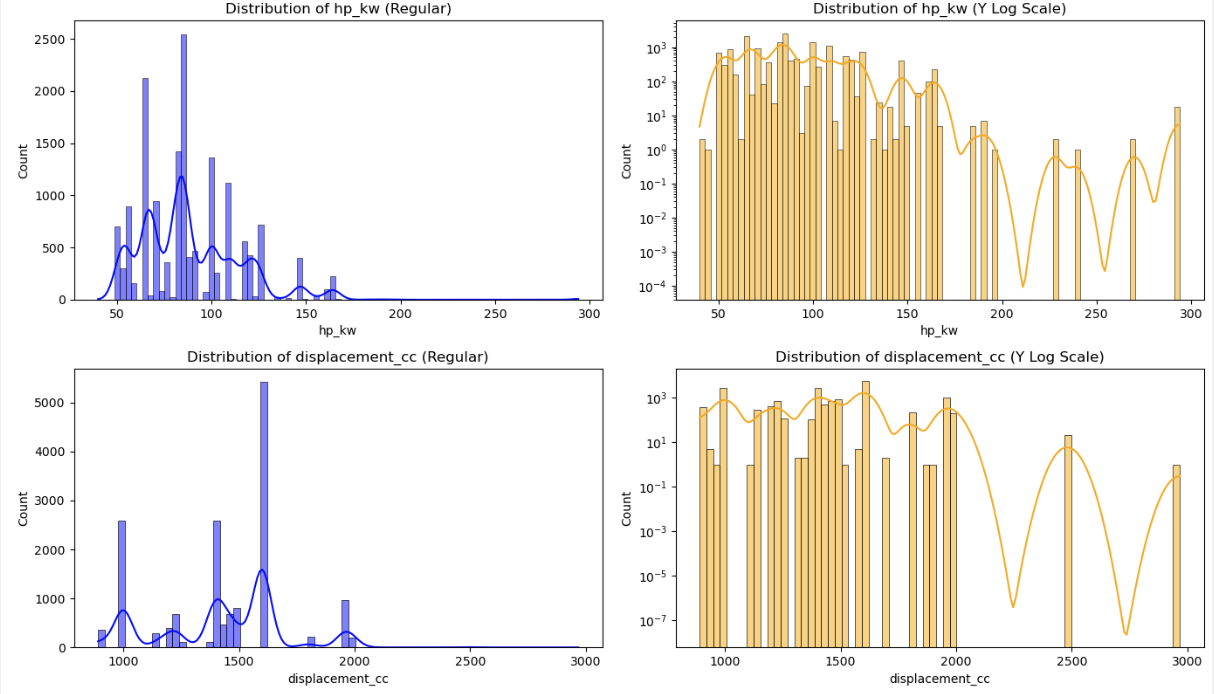
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Missing Count | Missing % | Impute Type | Action |
| make\_model | object | 0 | 0.0 | mode | none |
| age | float64 | 0 | 0.0 | median | none |
| drive\_chain | object | 0 | 0.0 | mode | none |
| weight\_kg | float64 | 0 | 0.0 | mean | none |
| displacement\_cc | float64 | 0 | 0.0 | mean | none |
| gearing\_type | object | 0 | 0.0 | mode | none |
| upholstery\_type | object | 0 | 0.0 | mode | none |
| paint\_type | object | 0 | 0.0 | mode | none |
| inspection\_new | int64 | 0 | 0.0 | mode | none |
| hp\_kw | float64 | 0 | 0.0 | mean | none |
| previous\_owners | float64 | 0 | 0.0 | mode | none |
| safety\_security | object | 0 | 0.0 | mode | none |
| body\_type | object | 0 | 0.0 | mode | none |
| extras | object | 0 | 0.0 | mode | none |
| entertainment\_media | object | 0 | 0.0 | mode | none |
| comfort\_convenience | object | 0 | 0.0 | mode | none |
| gears | float64 | 0 | 0.0 | mode | none |
| fuel | object | 0 | 0.0 | mode | none |
| type | object | 0 | 0.0 | mode | none |
| km | float64 | 0 | 0.0 | median | none |
| vat | object | 0 | 0.0 | mode | none |
| price | int64 | 0 | 0.0 | drop | none |
| cons\_comb | float64 | 0 | 0.0 | mean | none |

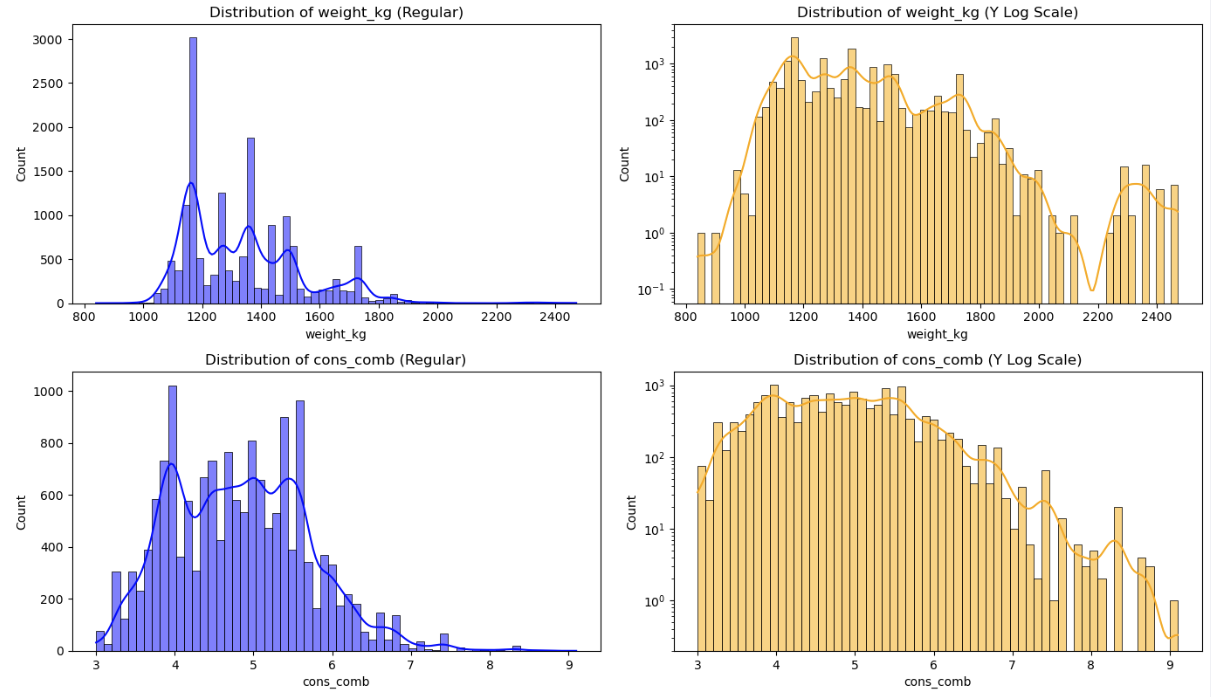
### Identify Numerical Predictors and Plot Frequency Distribution

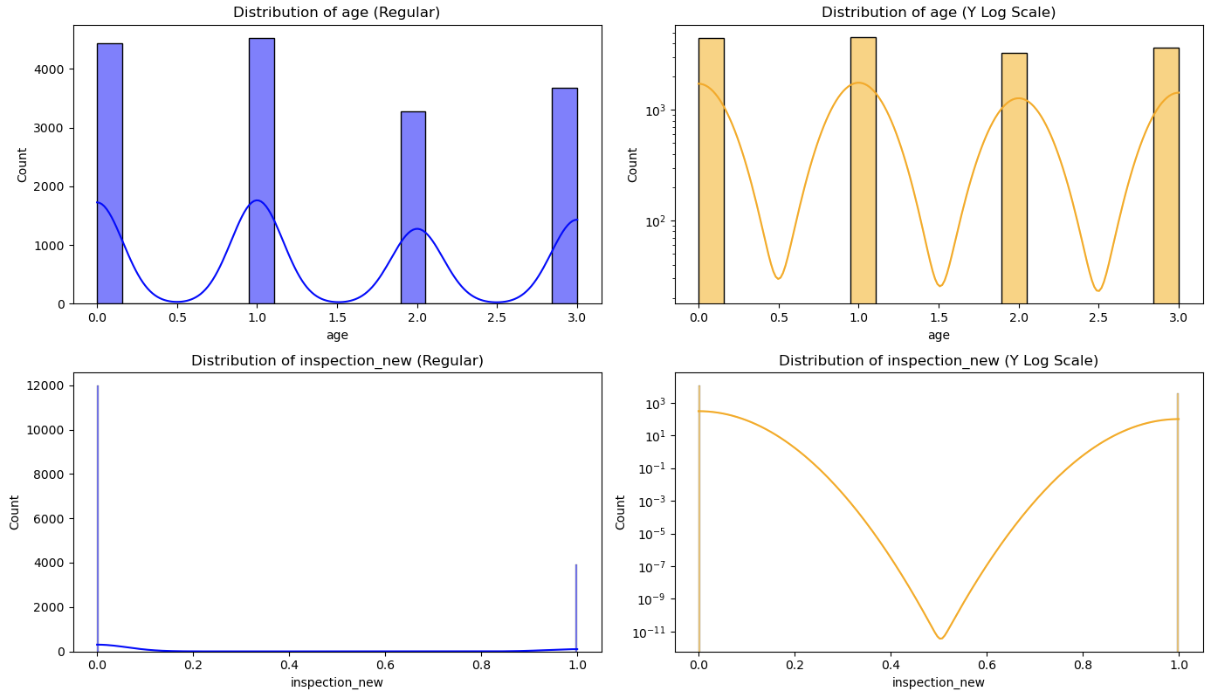
* Create 2 histogram plots per numeric column
  + First plot using regular yscale
  + Second plot using yscale = ‘log’





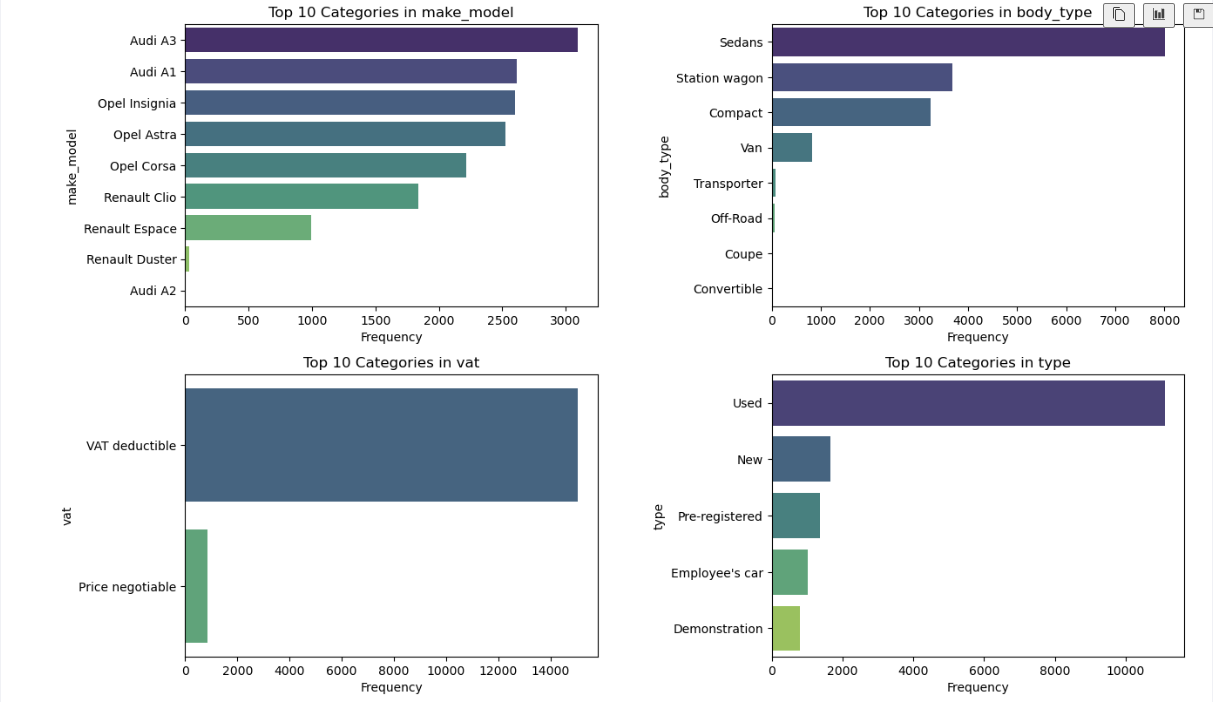


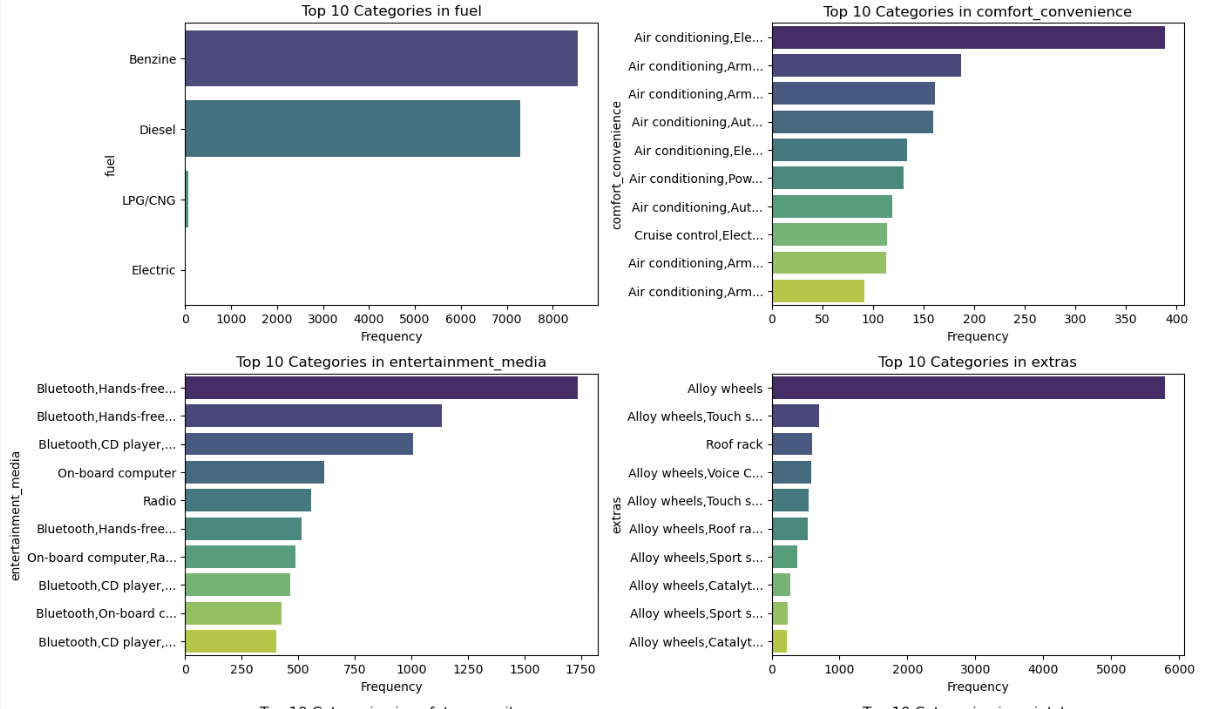


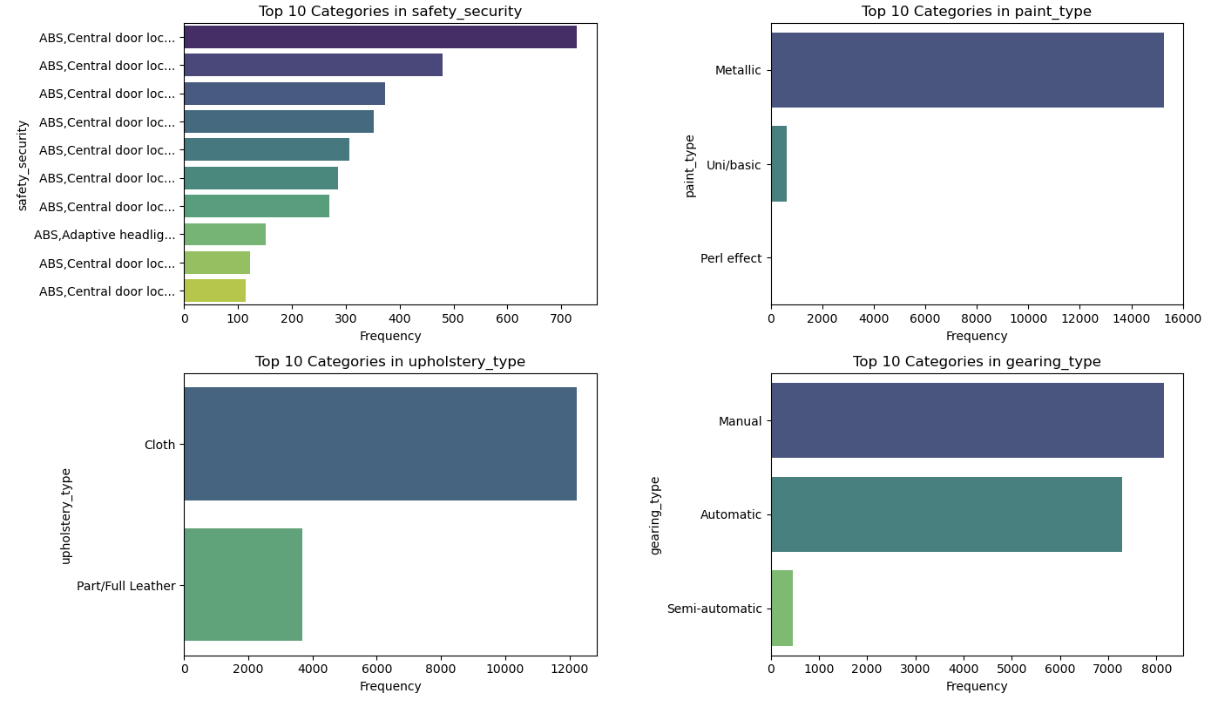


### Idendity Categorical Predictors and Plot Frequency Distributions

* Plot bar chart for each categorical column, with counts on x-axis & top-10 categories on the y-axis







A blue and white graph

AI-generated content may be incorrect.

* Note: The columns `["Comfort\_Convenience", "Entertainment\_Media", "Extras", "Safety\_Security"]` are all categorical columns with multiple discrete values in comma separated format in each cell. These will be treated with multi-label hot encoding later.

### Fix Columns with Low Frequency Values & Class Imbalances

* Strategy for ‘type’ column
  + Combine pre-registered and new cars into one
  + Combine employee car and demo car into one 🡪 low mileage/ low usage
* Strategy for all other categorical columns
  + Determine frequency % for each label value using value\_counts
  + If frequency > 5% for all labels, no change needed
  + If frequency <=5% for only 1 label, no change needed
  + If frequency <=5% for more than 1 labels, => combine low frequency values in the column into ‘Other’ label

---------------------------------------------------------------------------

Value counts for column 'type':

type

Used 0.697141

New 0.103613

Pre-registered 0.085705

Employee's car 0.063525

Demonstration 0.050016

Name: proportion, dtype: float64

Strategy for low frequency values in column 'type':

- Pre-registered → similar to New cars (low mileage, low usage) → combine with New

- Employee's car + Demonstration → short-term usage, small mileage → combine into Short-use

Post Processing:

---------------------------------------------------------------------------

Value counts for column 'type':

type

Used 0.697141

New 0.189318

Short-use 0.113541

Name: proportion, dtype: float64

---------------------------------------------------------------------------

Value counts for column 'body\_type':

body\_type

Sedans 0.502922

Station wagon 0.231040

Compact 0.203582

Van 0.051335

Transporter 0.005529

Off-Road 0.003519

Coupe 0.001571

Convertible 0.000503

Name: proportion, dtype: float64

Combining low frequency categories in 'body\_type' into 'Other'

Post-Processed Value counts for column 'body\_type':

body\_type

Sedans 0.502922

Station wagon 0.231040

Compact 0.203582

Van 0.051335

Other 0.011122

Name: proportion, dtype: float64

---------------------------------------------------------------------------

Value counts for column 'vat':

vat

VAT deductible 0.945272

Price negotiable 0.054728

Name: proportion, dtype: float64

No change needed as all values are significant in 'vat' with >5% frequency

---------------------------------------------------------------------------

Value counts for column 'fuel':

fuel

Benzine 0.537103

Diesel 0.458561

LPG/CNG 0.004021

Electric 0.000314

Name: proportion, dtype: float64

Combining low frequency categories in 'fuel' into 'Other'

Post-Processed Value counts for column 'fuel':

fuel

Benzine 0.537103

Diesel 0.458561

Other 0.004336

Name: proportion, dtype: float64

---------------------------------------------------------------------------

Value counts for column 'paint\_type':

paint\_type

Metallic 0.957964

Uni/basic 0.040025

Perl effect 0.002011

Name: proportion, dtype: float64

Combining low frequency categories in 'paint\_type' into 'Other'

Post-Processed Value counts for column 'paint\_type':

paint\_type

Metallic 0.957964

Other 0.042036

Name: proportion, dtype: float64

---------------------------------------------------------------------------

Value counts for column 'upholstery\_type':

upholstery\_type

Cloth 0.768709

Part/Full Leather 0.231291

Name: proportion, dtype: float64

No change needed as all values are significant in 'upholstery\_type' with >5% frequency

---------------------------------------------------------------------------

Value counts for column 'gearing\_type':

gearing\_type

Manual 0.512033

Automatic 0.458498

Semi-automatic 0.029469

Name: proportion, dtype: float64

No change needed as only one low frequency value in 'gearing\_type' with <=5% frequency

---------------------------------------------------------------------------

Value counts for column 'drive\_chain':

drive\_chain

front 0.986931

4WD 0.012818

rear 0.000251

Name: proportion, dtype: float64

Combining low frequency categories in 'drive\_chain' into 'Other'

Post-Processed Value counts for column 'drive\_chain':

drive\_chain

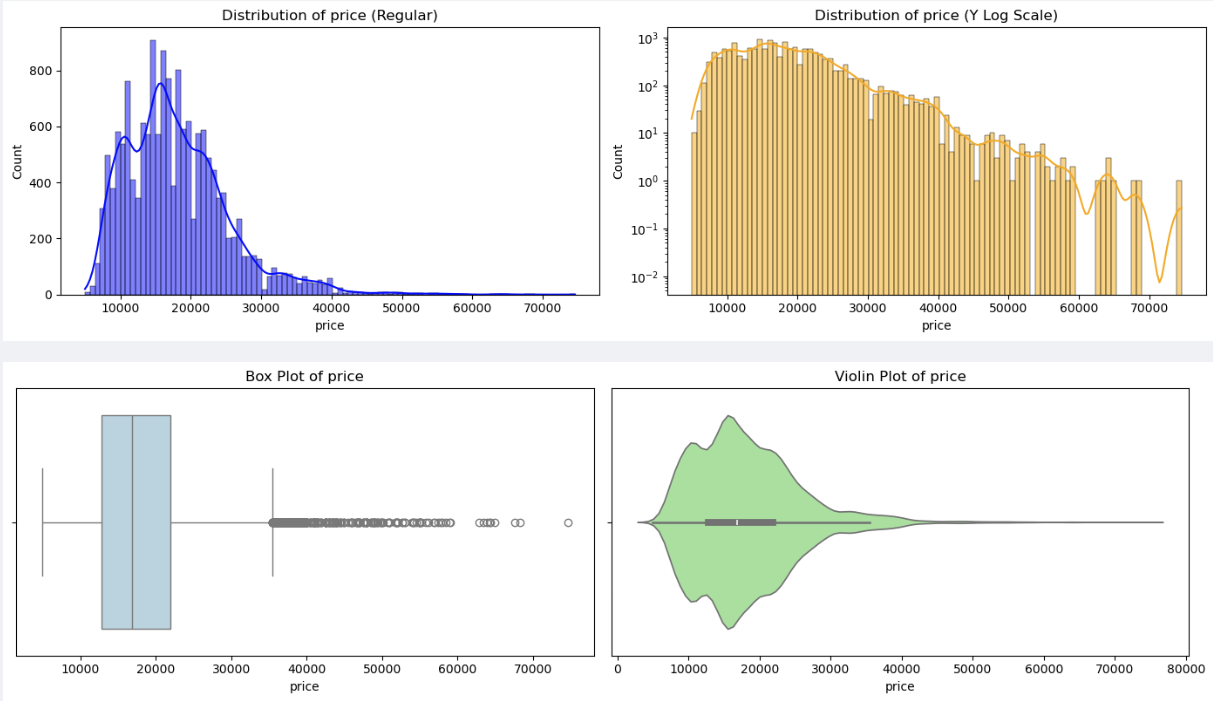
front 0.986931

Other 0.013069

Name: proportion, dtype: float64

### Identify Target Variable and Plot Frequency Distributions

* Plot histogram on ‘Price’ using normal and log y-scale
* Plot box plot and violin plot on ‘Price’
* If target variable is skewed, perform suitable outlier transformation
  + Calculate z-score for each row
  + Identify outliers with absolute(z-score) > threshold
  + Check original record count and % outliers before removing outliers
  + Delete the outliers
  + Check final record count after removing outliers
* Try with various values of threshold between 2-3.
* Check data quality of high price cars comparing with mean(price) for those models
* Plot histogram, box plot, violin plot on ‘price’ post outlier treatment



---------------------------------------------------------------------------

Original record count: 15755

Number of outliers in 'price' using Z-score method with threshold 3: 151 (0.96%)

Data shape after removing outliers: (15604, 24)

Records removed: 151

# Check for valid outliers for Audi AX and Renault Espace

Mean price for Audi A3: 20742.805790500977

Mean price for Renault Espace: 27865.379545454547



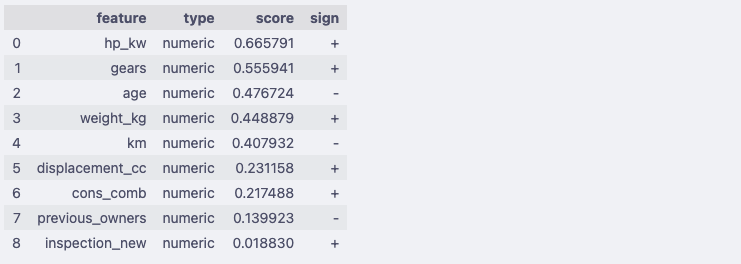
## Correlation Analysis

### Correlation Map Between Features and Target Variable

* Separate features and target into X and y
* Find correlation of numeric columns with target variable ‘price’
* Create a data frame with feature name, data type, abs(score), sign
* Plot a heatmap of numeric features correlation
* Display the numeric features with positive or negative high correlation with ‘price’

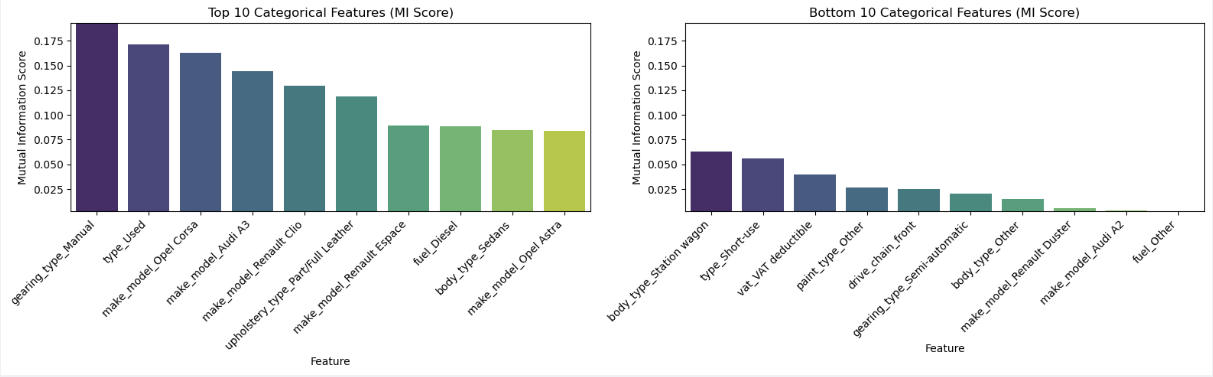
A chart of a heat map

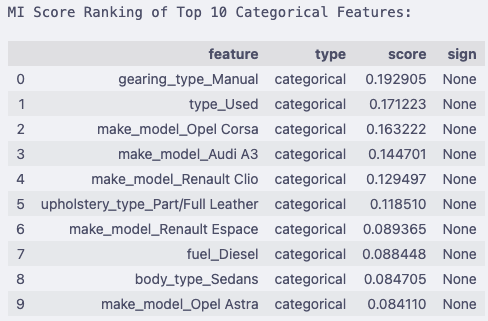
AI-generated content may be incorrect.

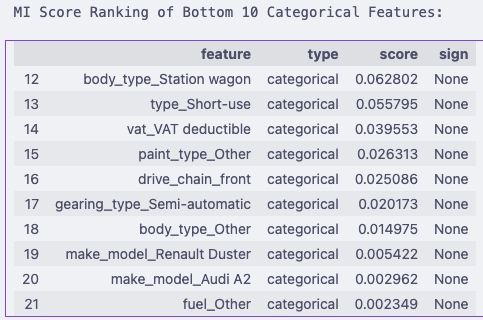


### Analyze Correlation Between Categorical Variable and Target Variable

* Separate features and target variables into X and y
* Select the list of categorical variables, excluding the ones that need multi-label hot encoding
* Make a copy of the data frame
* Apply one-hot encoding on the selected category variables
* Calculate Mutual Information (MI) scores of the category variables with target y
* Append results into data frame with columns : feature, type = ‘categorical’, score, sign = ‘none’
* Plot bar chart with feature on x-axis and score on y-axis, for Top-k features based on score
* Plot bar chart with feature on x-axis and score on y-axis, for Bottom-k features based on score







Combine the numeric and categorical features, and display the top-20 features with high correlation to target ‘price’.



#### Feature Importance Summary

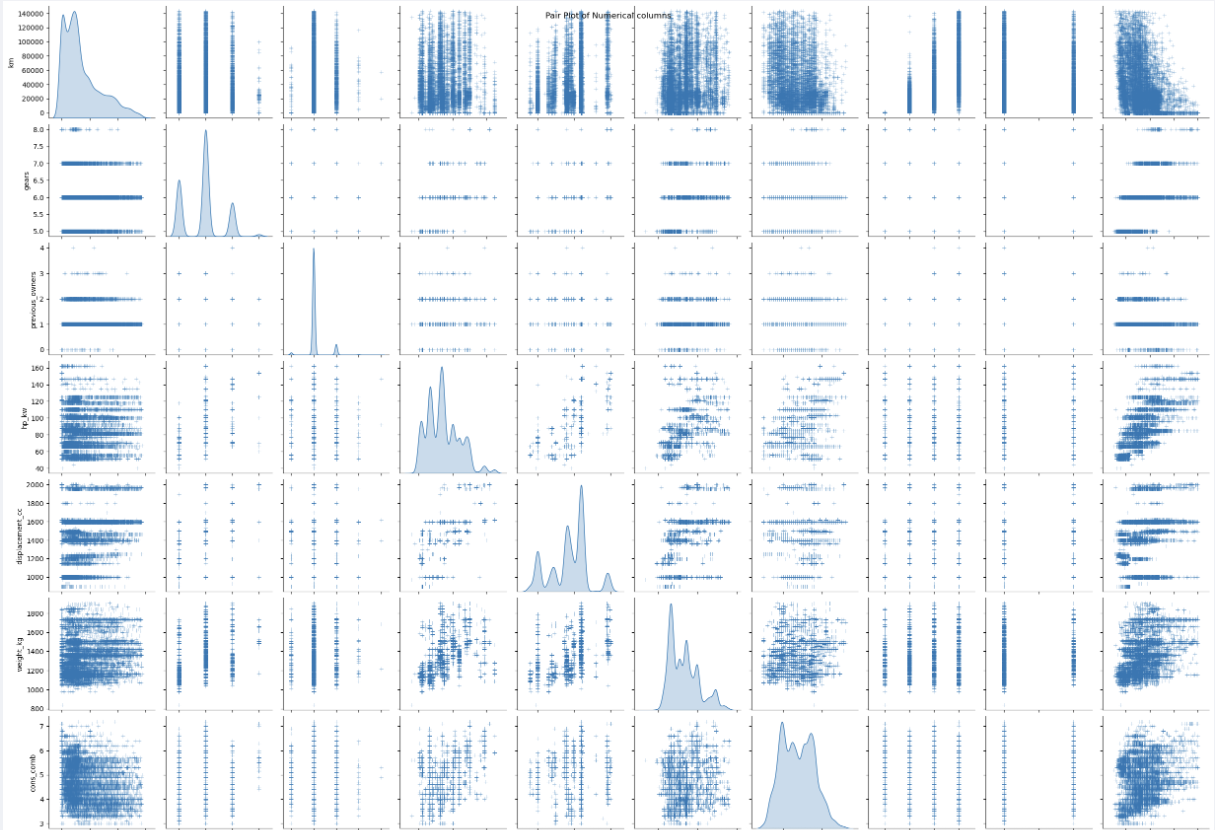
The table below combines the top numerical and categorical features most strongly associated with the target variable.

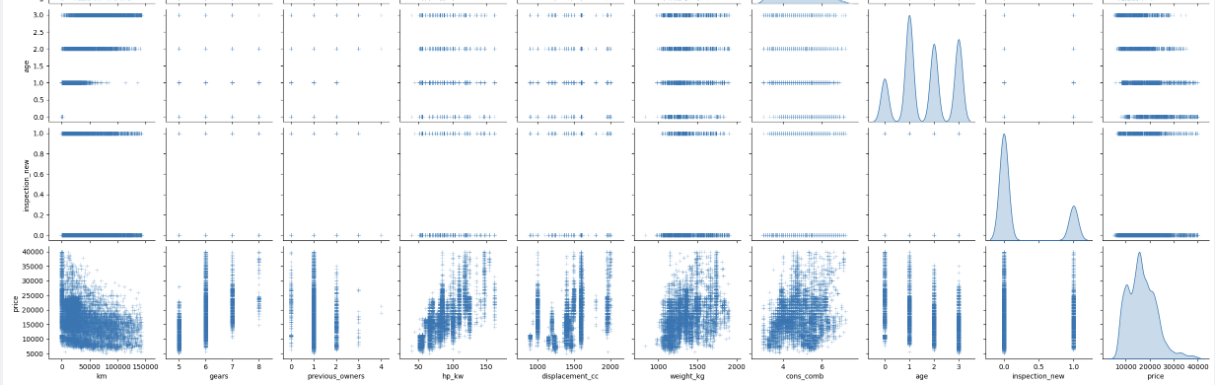
* **Numerical features** are ranked by absolute Pearson correlation with the target.
* **Categorical features** are ranked by mutual information (MI) with the target.
* This combined ranking helps prioritize which features to focus on for modeling and interpretation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rank | Feature | Type | Score | Sign | Inference |
| 1 | hp\_kw | Numeric | 0.666 | + | Strongest driver of price. More engine power → higher price. |
| 2 | gears | Numeric | 0.556 | + | More gears (modern transmissions) → generally higher prices. |
| 3 | age | Numeric | 0.477 | – | Older cars lose value. Strong negative effect. |
| 4 | weight\_kg | Numeric | 0.449 | + | Heavier cars (luxury/SUVs) are costlier. |
| 5 | km | Numeric | 0.408 | – | Higher mileage reduces price, though less than age. |
| 6 | displacement\_cc | Numeric | 0.231 | + | Bigger engines → higher prices, though weaker than horsepower. |
| 7 | cons\_comb | Numeric | 0.217 | + | Higher consumption → often premium/performance cars. |
| 8 | gearing\_type\_Manual | Categorical | 0.193 | – | Manual cars usually cheaper than automatics (signal of lower price). |
| 9 | type\_Used | Categorical | 0.171 | – | Used cars priced lower than new ones. |
| 10 | make\_model\_Opel Corsa | Categorical | 0.163 | – | Budget model → pulls price down. |
| 11 | make\_model\_Audi A3 | Categorical | 0.145 | + | Premium model → lifts price. |
| 12 | previous\_owners | Numeric | 0.140 | – | More owners reduce resale value. |
| 13 | make\_model\_Renault Clio | Categorical | 0.129 | – | Mass-market model, generally lower priced. |
| 14 | upholstery\_type\_Part/Full Leather | Categorical | 0.119 | + | Premium interiors increase price. |
| 15 | make\_model\_Renault Espace | Categorical | 0.089 | – | Family van, not premium → lower prices. |
| 16 | fuel\_Diesel | Categorical | 0.088 | ± | Diesel cars affect price; historically valuable in EU, now mixed. |
| 17 | body\_type\_Sedans | Categorical | 0.085 | ± | Sedans have moderate premium depending on market. |
| 18 | make\_model\_Opel Astra | Categorical | 0.084 | – | Mid-segment Opel → relatively lower priced. |
| 19 | make\_model\_Opel Insignia | Categorical | 0.079 | – | Slightly higher than Astra but still budget vs luxury brands. |
| 20 | body\_type\_Van | Categorical | 0.065 | – | Vans typically lower resale vs SUVs/sedans. |

#### Pair Plot for Correlation Analysis

* Create a pair plot across all numerical columns and analyze data

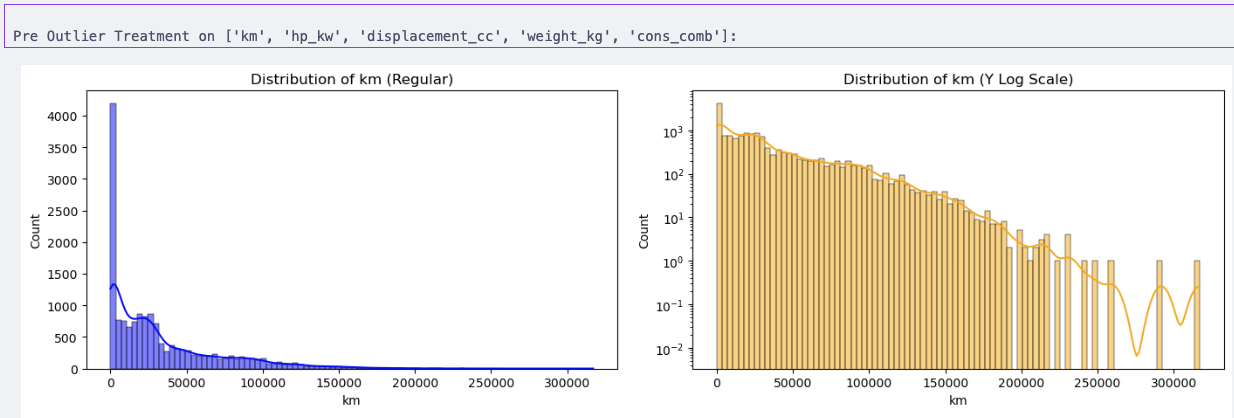


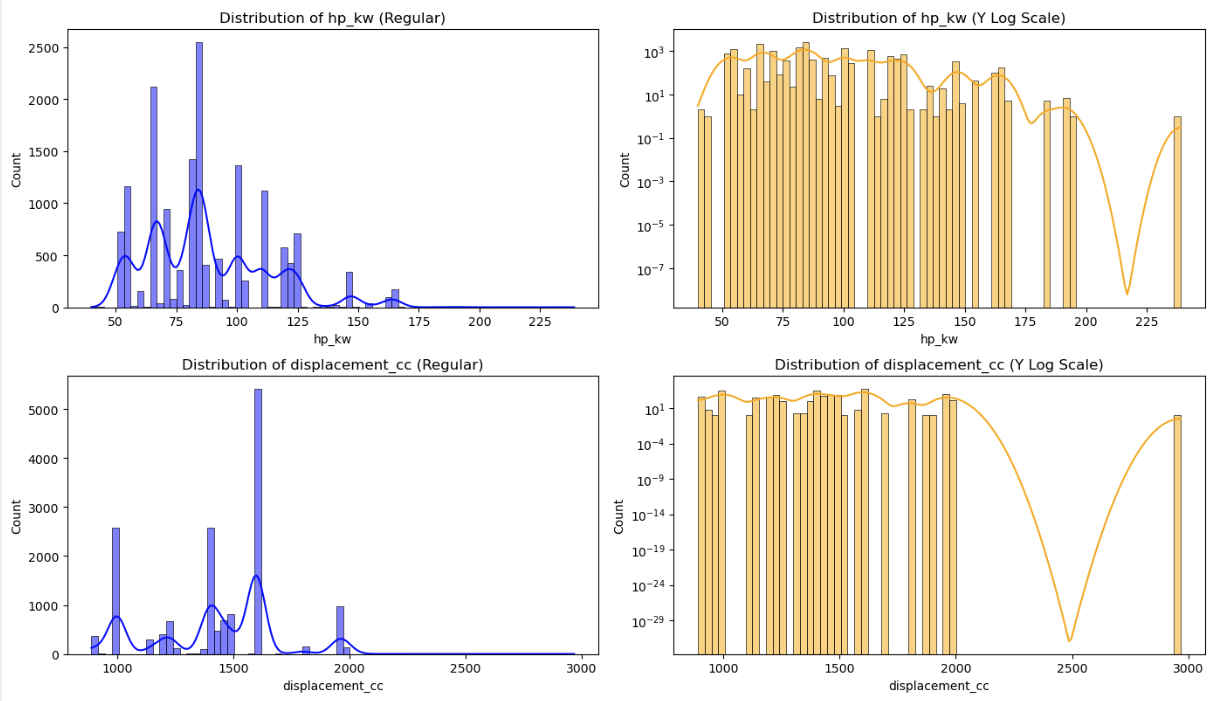


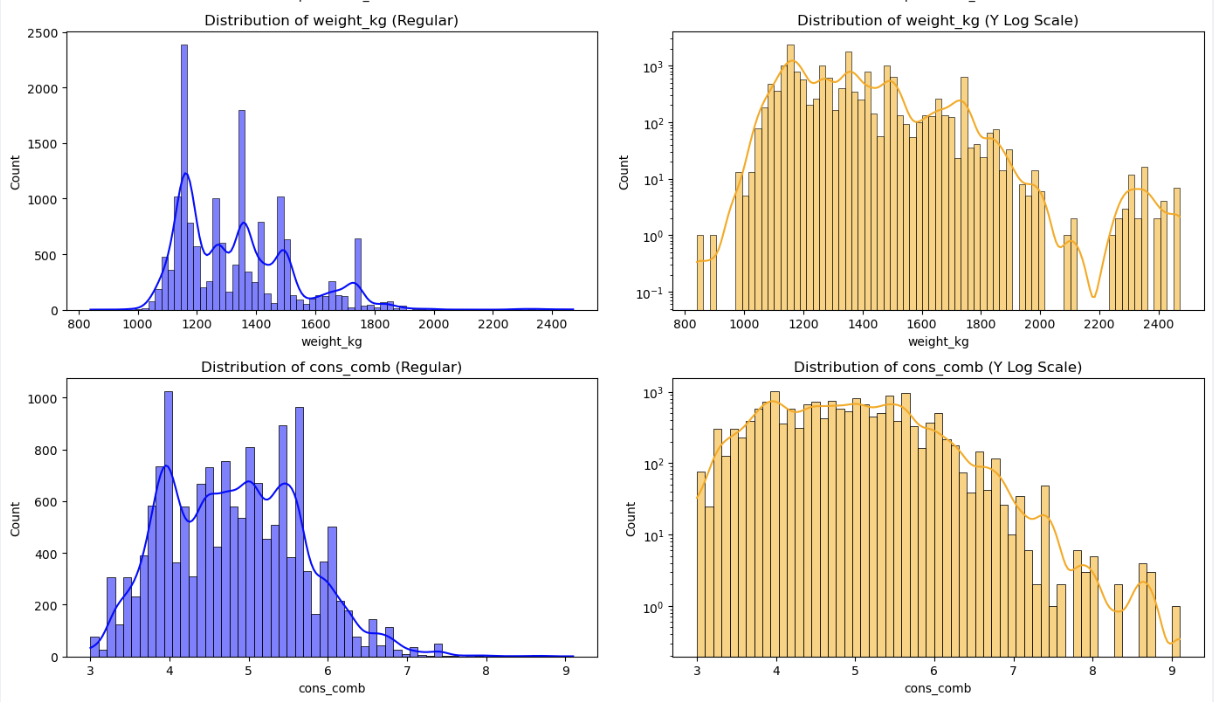
The pair plot confirms the correlations as identified earlier between numeric categorical variables and target ‘price’.

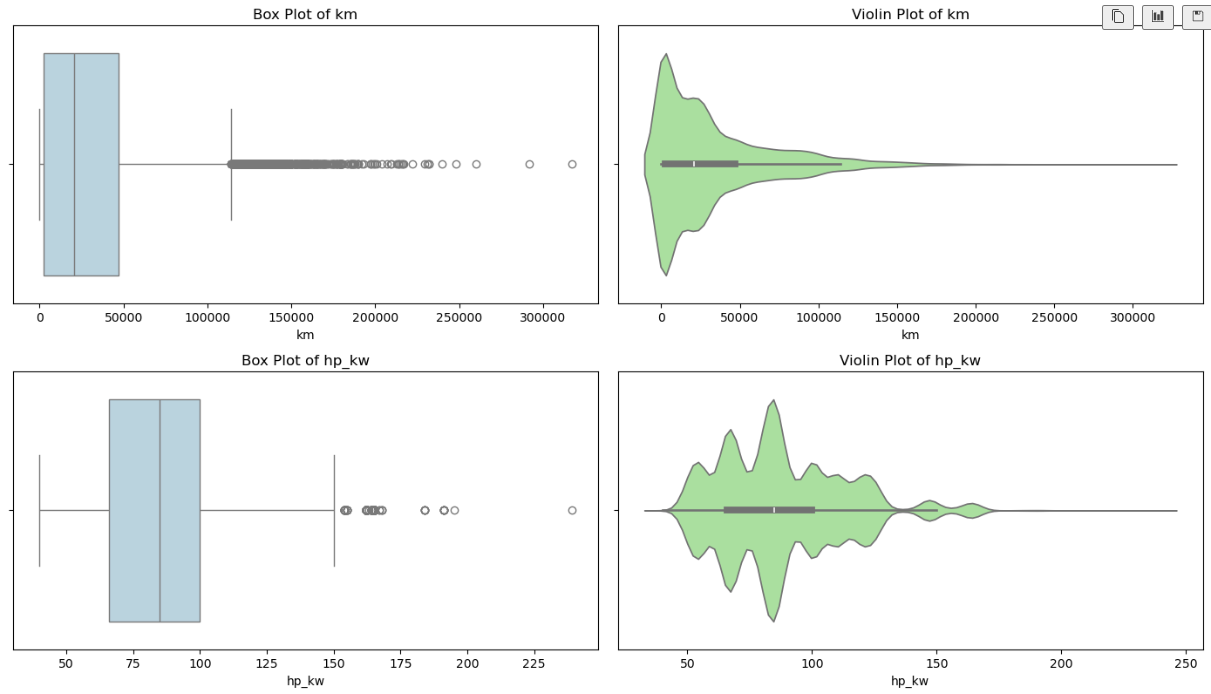
## Outlier Analysis

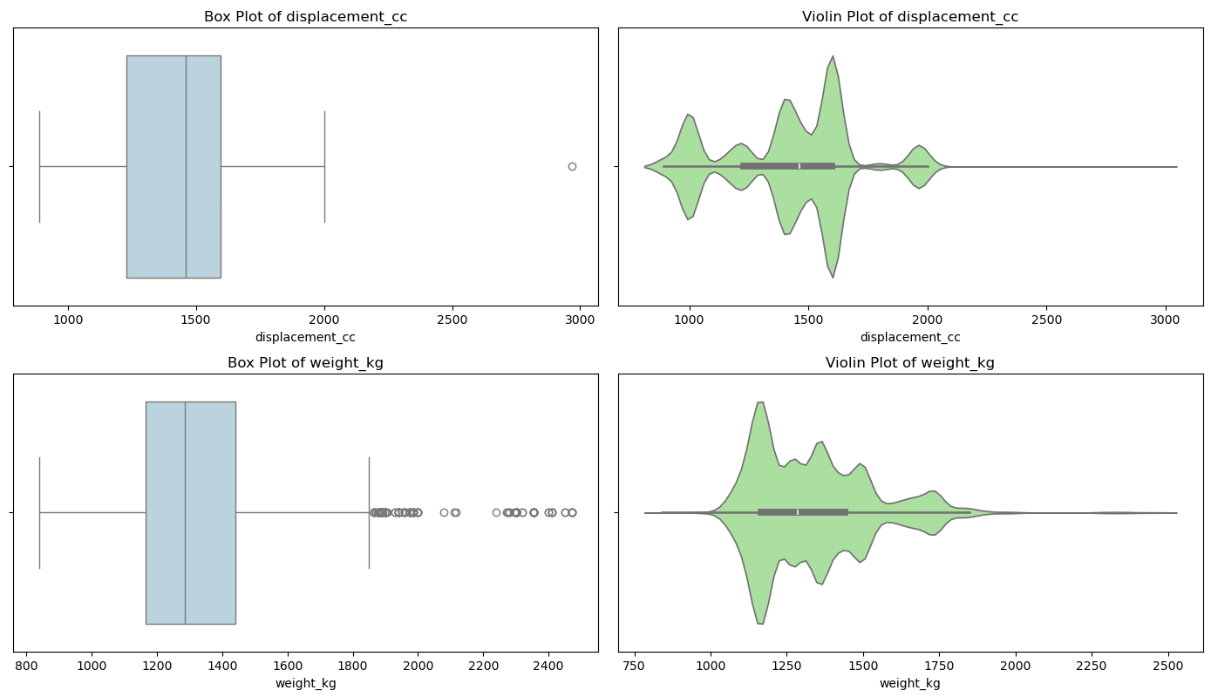
* Identify outliers in the numeric categorical variables
* Plot histogram with regular and log y-scale
* Plot box plot and violin plot











A green and grey graph

AI-generated content may be incorrect.

### Handle Outliers in Numeric Categorical Variables

* Use z-score functioned defined earlier to identify outliers using threshold between 2 and 3
* Remove outliers, which are beyond 3 sigma deviation
* Plot histograms, box and violin charts to validate outcomes

---------------------------------------------------------------------------

Original record count: 15755

Number of outliers in 'km' using Z-score method with **threshold 3: 260 (1.65%)**

Data shape after removing outliers: (15495, 24)

Records removed: 260

---------------------------------------------------------------------------

Original record count: 15495

Number of outliers in 'hp\_kw' using Z-score method with **threshold 3: 189 (1.22%)**

Data shape after removing outliers: (15306, 24)

Records removed: 189

---------------------------------------------------------------------------

Original record count: 15306

Number of outliers in 'displacement\_cc' using Z-score method with **threshold 3: 0 (0.00%)**

Data shape after removing outliers: (15306, 24)

Records removed: 0

---------------------------------------------------------------------------

Original record count: 15306

Number of outliers in 'weight\_kg' using Z-score method with **threshold 3: 83 (0.54%)**

Data shape after removing outliers: (15223, 24)

Records removed: 83

---------------------------------------------------------------------------

Original record count: 15223

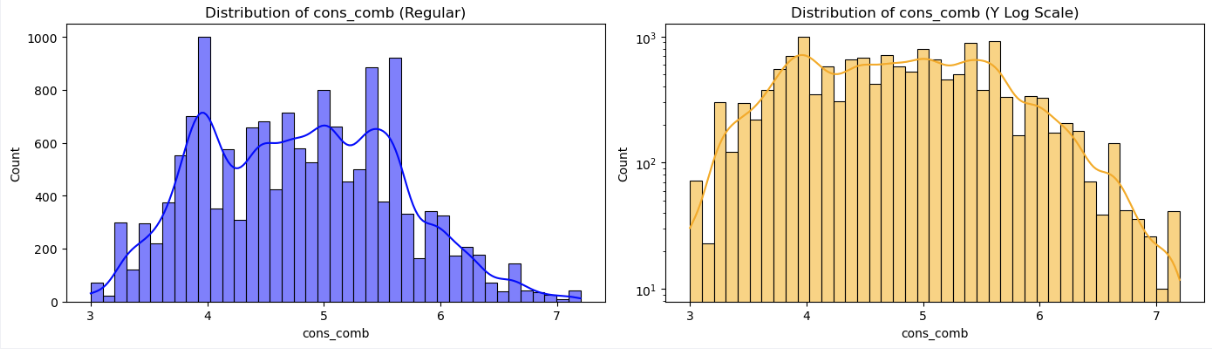
Number of outliers in 'cons\_comb' using Z-score method with **threshold 3: 17 (0.11%)**

Data shape after removing outliers: (15206, 24)

Records removed: 17







A collage of graphs

AI-generated content may be incorrect.

A diagram of a graph

AI-generated content may be incorrect.

A green and black graph

AI-generated content may be incorrect.

#### Data Quality Checks on Remaining Outliers

* Check if there are cars with age zero, but not new or high km
* Check if there are cars with km zero, but not new or high age
* Clean the outlier records
* Check for realistic value range for desired models for columns weight, displacement, horsepower, fuel consumption, age, mileage
* Check for any missing values, negative values, zero values and treat them
* Check for outliers using quantile and see if these are in small range (we have already applied z-score method for outlier removal)

---------------------------------------------------------------------------

Data Quality Checks Driven Cleanup - age

- There are 4203 cars with zero age which is valid for new cars.

- There are 1797 cars with zero age and high km.

- There are 1494 cars with zero age and type <> new.

- There are 862 cars with zero age, high km and type <> new.

- These records are likely data quality issues and will be removed.

Data Quality Counts Post Cleanup - age

- There are 1774 cars with zero age which is valid for new cars.

- There are 0 cars with zero age and high km.

- There are 0 cars with zero age and type <> new.

- There are 0 cars with zero age, high km and type <> new.

- These records are likely data quality issues and will be removed.

---------------------------------------------------------------------------

Data Quality Checks Driven Cleanup - km

- There are 19 cars with zero km which is valid for new cars.

- There are 0 cars with zero km and type <> new.

- There are 0 cars with zero km, age <> 0.

- These look like brand new cars, example the ones in showroom without usage, hence will be retained.

---------------------------------------------------------------------------

Checking for realistic value ranges for ['Audi A1', 'Audi A2', 'Audi A3', 'Opel Astra', 'Opel Corsa','Opel Insignia', 'Renault Clio', 'Renault Espace']

- Car weight should be positive

- Engine displacement should be positive

- Car weight should be between 600 and 3500 kg

- Displacement should be between 890 and 3000 cc

- Engine power (hp\_kw) should be between 40 and 300 kW

- Combined fuel consumption (cons\_comb) should be between 3 and 15 L/100km

- Car age should be between 0 and 30 years

- Car mileage (km) should be between 0 and 400,000 km

---------------------------------------------------------------------------

Post Outlier Treatment Summary Statistics (Min, Max, Missing, Negative, Zero counts):

- weight\_kg: min=840.0, max=1905.0, missing=0, negative=0, zero=0

- displacement\_cc: min=898.0, max=2000.0, missing=0, negative=0, zero=0

- hp\_kw: min=40.0, max=162.0, missing=0, negative=0, zero=0

- cons\_comb: min=3.0, max=7.2, missing=0, negative=0, zero=0

- age: min=0.0, max=3.0, missing=0, negative=0, zero=1774

- km: min=0.0, max=143258.0, missing=0, negative=0, zero=19

---------------------------------------------------------------------------

Post Z-score Outlier Treatment - Records above 99th percentile:

- Weight (kg) above 99th percentile: 128

- Displacement (cc) above 99th percentile: 80

- HP (kW) above 99th percentile: 116

- Fuel consumption (cons\_comb) above 99th percentile: 98

---------------------------------------------------------------------------

Post Outlier Treatment and Data Validation:

- All values of weight\_kg, displacement\_cc, hp\_kw, cons\_comb, age, km are within realistic ranges

- No missing or negative values in key columns weight\_kg, displacement\_cc, hp\_kw, cons\_comb, age, km

- There are few records above 99th percentile in Weight, Displacement, HP, fuel consumption which are valid high-end cars

## Feature Engineering

### Fix Redundant Columns and Create New Ones

* While there are few highly correlated columns that can be removed, we will retain them for now and use Ridge and Lasso regression later
* Apply one-hot encoding on categorical variables using pd.get\_dummies
* For the new columns generated, convert the boolean data type to integer

<class 'pandas.core.frame.DataFrame'>

Index: 12777 entries, 0 to 15913

Data columns (total 35 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 price 12777 non-null int64

1 km 12777 non-null float64

2 gears 12777 non-null float64

3 comfort\_convenience 12777 non-null object

4 entertainment\_media 12777 non-null object

5 extras 12777 non-null object

6 safety\_security 12777 non-null object

7 age 12777 non-null float64

8 previous\_owners 12777 non-null float64

9 hp\_kw 12777 non-null float64

10 inspection\_new 12777 non-null int64

11 displacement\_cc 12777 non-null float64

12 weight\_kg 12777 non-null float64

13 cons\_comb 12777 non-null float64

14 make\_model\_Audi A2 12777 non-null int64

15 make\_model\_Audi A3 12777 non-null int64

16 make\_model\_Opel Astra 12777 non-null int64

17 make\_model\_Opel Corsa 12777 non-null int64

18 make\_model\_Opel Insignia 12777 non-null int64

19 make\_model\_Renault Clio 12777 non-null int64

...

33 gearing\_type\_Semi-automatic 12777 non-null int64

34 drive\_chain\_front 12777 non-null int64

dtypes: float64(8), int64(23), object(4)

### Analysis & Feature Engineering on Multi-Label Variables

* Analyze ['Comfort\_Convenience', 'Entertainment\_Media', 'Extras', 'Safety\_Security'] columns
* Check unique values in each feature spec column
* For each multi-label categorical column:
  + Split values by comma, convert to a pandasSeries where each row is a list of features
  + Flatten all lists into a signle list all\_features, to count occurrences of each feature
  + Keep only features present in >= threshold\_percent of rows

comfort\_convenience: 26 features retained out of 38 total

entertainment\_media: 9 features retained out of 10 total

extras: 8 features retained out of 17 total

safety\_security: 27 features retained out of 29 total

List of commonly used features retained in the columns:

{'comfort\_convenience': ['Air conditioning',

'Armrest',

'Automatic climate control',

'Cruise control',

'Electrical side mirrors',

'Hill Holder',

'Leather steering wheel',

'Light sensor',

'Multi-function steering wheel',

'Navigation system',

'Park Distance Control',

'Parking assist system sensors rear',

'Power windows',

'Rain sensor',

'Seat heating',

'Start-stop system',

'Lumbar support',

'Tinted windows',

'Parking assist system sensors front',

'Split rear seats',

'Keyless central door lock',

'Electrically heated windshield',

'Parking assist system camera',

'Electrically adjustable seats',

'Electric tailgate',

'Heated steering wheel'],

'entertainment\_media': ['Bluetooth',

'Hands-free equipment',

'On-board computer',

'Radio',

'Sound system',

'MP3',

'CD player',

'USB',

'Digital radio'],

'extras': ['Alloy wheels',

'Catalytic Converter',

'Voice Control',

'Sport seats',

'Sport suspension',

'Sport package',

'Touch screen',

'Roof rack'],

'safety\_security': ['ABS',

'Central door lock',

'Daytime running lights',

'Driver-side airbag',

'Electronic stability control',

'Fog lights',

'Immobilizer',

'Isofix',

'Passenger-side airbag',

'Power steering',

'Side airbag',

'Tire pressure monitoring system',

'Traction control',

'Xenon headlights',

'Central door lock with remote control',

'Head airbag',

'Alarm system',

'Emergency system',

'LED Headlights',

'Adaptive headlights',

'LED Daytime Running Lights',

'Rear airbag',

'Emergency brake assistant',

'Adaptive Cruise Control',

'Traffic sign recognition',

'Lane departure warning system',

'Blind spot monitor']}

### Perform Multi-Label Feature Encoding

* For each multi-label categorical column:
  + Split values by comma, convert to a pandasSeries where each row is a list of features
  + Multi-hot encode using only common features identified in previous steps
  + Concatenate encoded columns and drop original

Encoded DataFrame shape: (12777, 101)

<class 'pandas.core.frame.DataFrame'>

Index: 12777 entries, 0 to 15913

Columns: 101 entries, price to safety\_security\_Blind spot monitor

dtypes: float64(8), int64(93)

memory usage: 9.9 MB

### Split Data into Training and Test

* Separate dependent and independent variables into X and y
* Create X\_train, X\_test, y\_train, y\_test dataframes using test size 0.2 and random state 42 (for consistency across runs)

Shape of X\_train: (10221, 100), y\_train: (10221,), X\_test: (2556, 100), y\_test: (2556,)

### Scale Features

#### Recommended workflow:

* Split data into train and test sets.
* Fit the scaler (e.g., StandardScaler, MinMaxScaler) only on the training data.
* Transform both train and test sets using the scaler fitted on the training data.
* Use the scaled data for all your models.
* Why?
  + Scaling before splitting can cause data leakage, leading to overly optimistic model performance.
  + Scaling after splitting ensures fair and realistic evaluation for all models.

#### When to use StandardScaler vs MinMaxScalar:

* Standard
  + Most features are continuous and roughly bell-shaped (normal distribution).
  + We are using models that assume standardized data (linear regression, logistic regression, SVM, etc.).
  + If we want to handle outliers more robustly (StandardScaler is less sensitive than MinMaxScaler).
* MinMaxScaler:
  + Your features have a known, fixed range and you want to scale everything to [0, 1].
  + You are using models that are sensitive to the scale of input (e.g., neural networks).
  + Your data does not have extreme outliers.

#### Scale Features

* Select StandardScalar() approach
* Fit scalar transform using X\_train dataset
* Transform X\_train and y\_train using the fitted scalar

Shape of X\_train\_scaled, X\_test\_scaled : (10103, 99), (2526, 99)

# Linear Regression Models

## Baseline Linear Regression Model

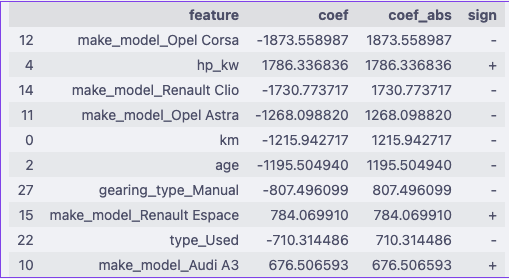
### Build Basic Linear Regression Model & Evaluate Performance

* Initialize LinearRegression() model
* Fit the model on X\_train\_scaled dataset
* Predict y\_train\_pred, y\_test\_pred using the fitted linear regression model
* Create a lr\_coeff dataframe with features, coefficients, absolute(coefficients), sign
* Display top-N coefficients using absolute values
* Analyze model scores for both Train and Test data –
  + R2, MSE, RMSE, MAE, MAPE using y\_actual and y\_predicted values
  + Variance, Standard Deviation, Mean using y\_actual values

---------------------------------------------------------------------------

Top 10 Coefficients with sign

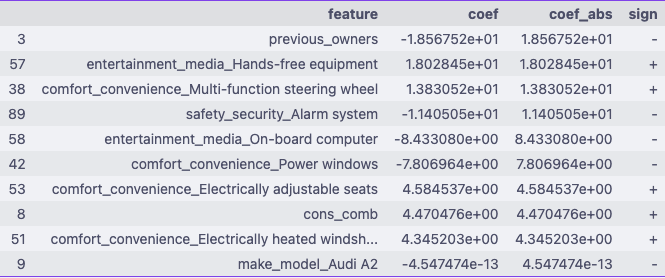
---------------------------------------------------------------------------



---------------------------------------------------------------------------

Bottom 10 Coefficients with sign

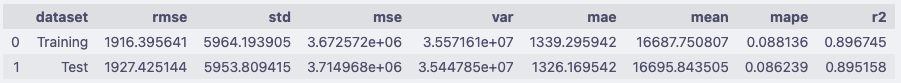
---------------------------------------------------------------------------



---------------------------------------------------------------------------

Analyze Model Performance – Baseline Linear Regression

---------------------------------------------------------------------------



#### Analysis of baseline model

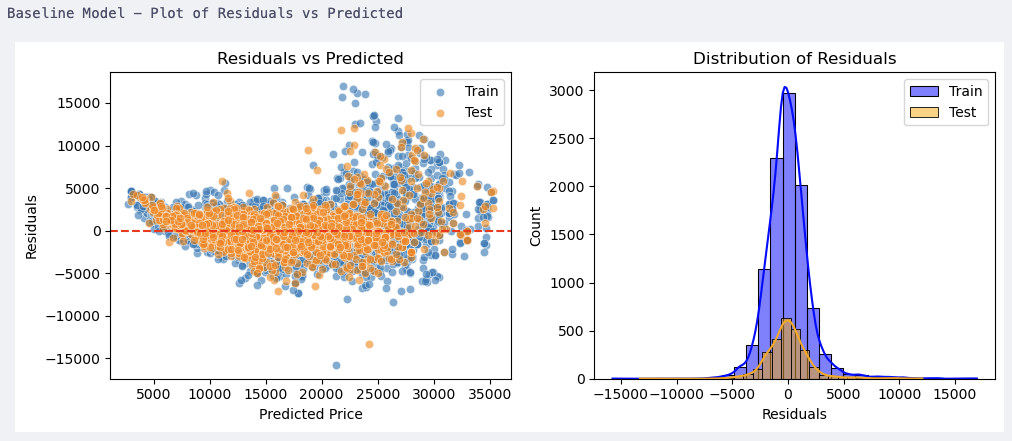
* **rmse < std: Good**. Our model’s error is less than the standard deviation of the actual values, meaning it’s better than just predicting the mean.
* **mse < var for train and test: Good**. The model’s mean squared error is less than the variance of the actuals, indicating it’s capturing signal.
* **mae < mean for train and test: Good**. The average error is much less than the average value, so your predictions are reasonably close.
* **r2 > 0.89 for both and close: Very good**. High and similar R² on train and test means your model is both accurate and generalizes well.

#### Summary:

* Model is performing well overall.
* Keep an eye on the gap between train and test metrics; if it grows, consider regularization or more data.

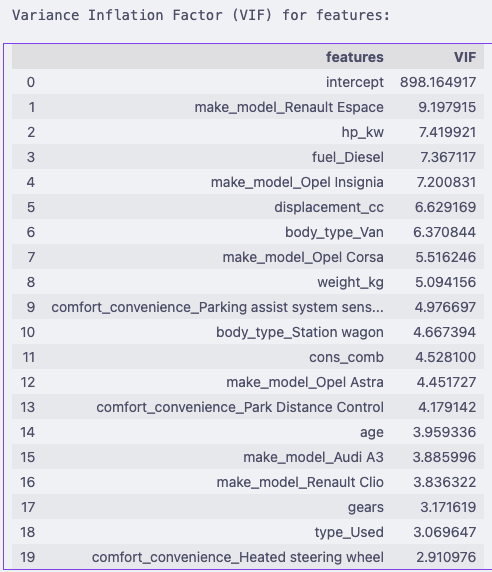
### Analyze Residuals

* Find residuals for trained data (y\_train – y\_train\_pred)
* Find residuals for test data (y\_test – y\_test\_pred)
* Residuals vs Predicted targets values chart
  + Scatter plot of y predicted values vs residuals of trained data
  + Overlay scatter plot of y predicted values vs residuals of test data
* Residuals distribution chart
  + Histogram of residuals of trained data
  + Overlay histogram with residuals of test data



### Check Multicollinearity using Variance Inflation Factor (VIF)

* Create a DataFrame to store VIF results.
* Add a column intercept using a constant value
* For each feature (including the constant intercept), calculates the Variance Inflation Factor (VIF) using variance\_inflation\_factor function.
* VIF measures how much a feature is linearly predicted by the other features (i.e., multicollinearity).
* How VIF works:
  + For each feature, it fits a regression model predicting that feature from all the others.
  + VIF = 1 / (1 - R²) from that regression.
  + High VIF (>5 or >10) means the feature is highly collinear with others



#### Interpretation

- VIF < 5 → Fine

- 5 ≤ VIF < 10 → Moderate multicollinearity, keep an eye

- VIF ≥ 10 → Strong multicollinearity, consider fixing

|  |  |  |
| --- | --- | --- |
| Feature | VIF | Interpretation |
| intercept | 898 | Ignore. Intercept often has artificially high VIF and doesn’t affect feature decisions. |
| make\_model\_Renault Espace | 9.19 | High. Likely overlaps with body\_type (Van) and maybe weight\_kg. |
| hp\_kw | 7.41 | High. Horsepower is highly correlated with displacement\_cc and weight\_kg. |
| fuel\_Diesel | 7.36 | High. Fuel type interacts with body\_type and consumption (cons\_comb). |
| make\_model\_Opel Insignia | 7.20 | High. Strongly collinear with other Opel models + body types. |
| displacement\_cc | 6.62 | Multicollinear with hp\_kw (both capture engine size/power). |
| body\_type\_Van | 6.37 | Overlaps with Renault Espace (most Espace are vans) → redundancy. |
| make\_model\_Opel Corsa | 5.51 | Moderate collinearity, probably with other Opel models. |
| weight\_kg | 5.10 | Correlated with hp\_kw and displacement\_cc. Bigger engines = heavier cars. |
| comfort\_convenience\_Parking assist… | 4.98 | Moderate. Likely overlaps with other comfort features (Park Distance Control). |
| body\_type\_Station wagon | 4.66 | Overlaps with specific models (Opel Astra wagon, Renault Espace). |
| cons\_comb | 4.52 | Correlated with engine size (displacement\_cc, hp\_kw). |
| make\_model\_Opel Astra | 4.45 | Model dummies always show collinearity with body\_type & brand group. |
| comfort\_convenience\_Park Distance Control | 4.18 | Redundant with Parking assist system sensors. |
| age | 3.96 | Low–moderate. Correlates slightly with km (which you probably dropped earlier due to multicollinearity). |
| make\_model\_Audi A3 | 3.89 | Fine, moderate. Captures brand-specific pricing. |
|  |  |  |
| make\_model\_Renault Clio | 3.83 | Acceptable. Still collinear with Renault brand dummies. |
| gears | 3.17 | Acceptable. Correlates with hp\_kw (higher performance cars → more gears). |
| type\_Used | 3.06 | Acceptable. Correlates slightly with age and km. |
| comfort\_convenience\_Rain sensor | 2.91 | Low, no issue. |

## Ridge Regression Implementation

### Define Alpha Values

* We will start with alpha range of alpha\_range = [0.01, 0.1, 1, 10, 50, 100, 200, 300, 400, 500]

### Apply Ridge Regression, Analyze Performance & Find Best Alpha Value

#### Apply Ridge Regression

* Function to fit a RidgeCV model to the training data and returns predictions for both train and test sets. Also returns the chosen alpha value
* Use RidgeCV with *cv=KFold(n\_splits=5, shuffle=True, random\_state=42)*
  + *cv=5 is same as default KFold(n\_splits=5, shuffle=False)*
  + *cv=KFold(n\_splits=5, shuffle=True, random\_state=42) => This explicitly creates a KFold cross-validator with 5 splits,*
  + *shuffle=True randomizes the order of the data before splitting, especially needed if your dataset is ordered*
  + *random\_state=42 ensures reproducibility (the same splits every time you run)*
* Fit the model on trained data
* Predict the target values on train and test data both
* Use this model to identify the chosen alpha, which will later be used for narrowing the range

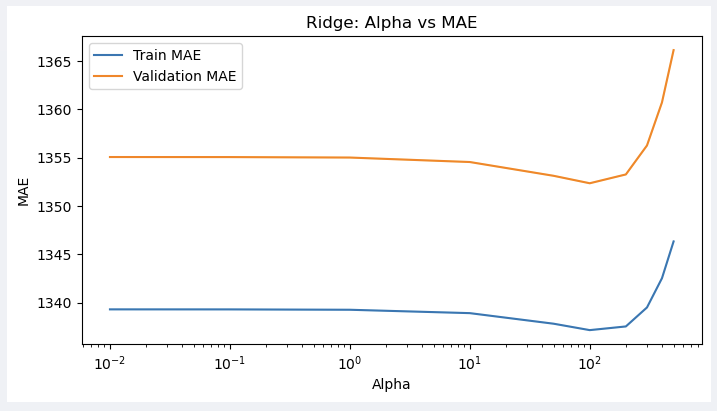
#### Analyze Model Performance & Identify Chosen Alpha

* Use the reusable function defined earlier to calculate and compare the model metrics for train and test data
* Use the reusable function defined earlier to chart
  + Residuals vs Predicted plot
  + Distribution of Residuals



#### Plot Error-Alpha chart

* Since RidgeCV does not store the scores for each CV fold, we will calculate this using Ridge() and save the train and test scores of each run of alpha
  + Initialize Ridge() model
  + Fit the model on training data
  + Apply the model on training and test data to predict training scores and validation scores
  + Apply cross validation score method using “neg\_mean\_absolute\_error”
  + Plot the line graph of alpha on x-axis vs training/ testing scores on y-axis



#### Find Chosen Alpha Value

* Identify Chosen alpha value using ridge.coef\_ identified in 3.2.2.1
* This is based on RidgeCV model outcome

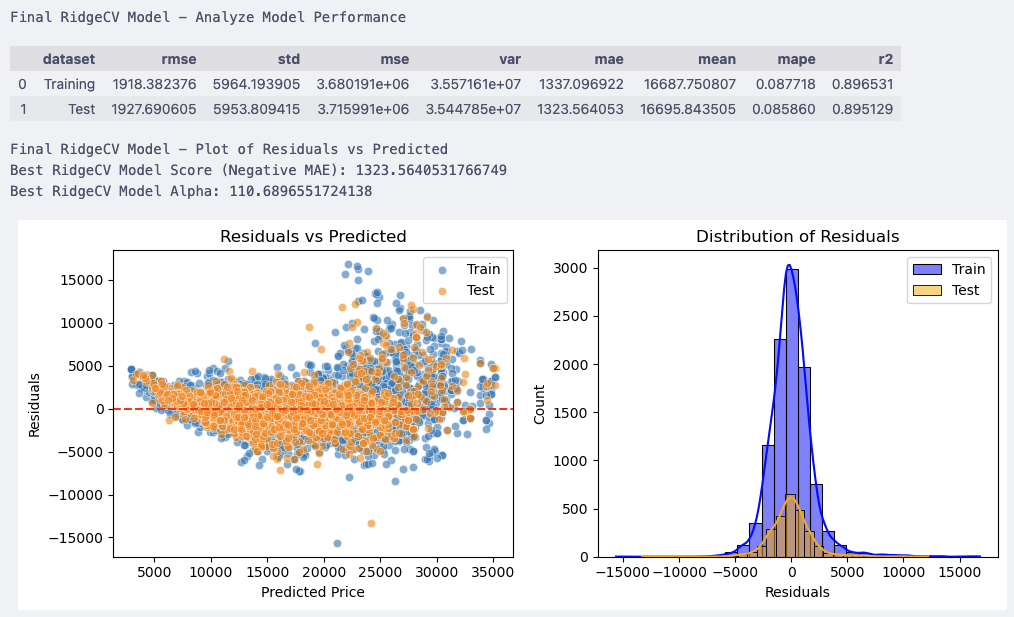
---------------------------------------------------------------------------

Chosen alpha (Ridge): 100.0

---------------------------------------------------------------------------

### Fine Tune Alpha, Analyze Performance, Error-Alpha Chart, Coefficients

* We will narrow down the alpha range around the chosen alpha value from 3.2.2.4
* narrow\_alpha\_range = np.*linspace*(90, 120, 30) # 30 values between 90 and 120
* Run the resusable function ridgecv model using the narrow alpha range
* Analyze the model performance on train and test data
  + Plot residuals vs predicted chart
  + Plot distribution
* Find the best ridgeCV alpha value for Test data and corresponding “mean\_absolute\_error”
* Using the reusable function defined earlier, plot the error-alpha chart.
* Show the coefficients for top and bottom k features
  + Features: X\_train.columns
  + Coef: ridge.coef\_
  + Coef\_abs : absolute value of ridge.coef\_
  + Sign : + or -ve based on Coef



A graph with numbers and lines

AI-generated content may be incorrect.

Since the alpha range is narrow, the MAE line is almost flat.

--------------------------------------------------------------------------

Final Ridge Model Coefficients with best score alpha=110.6896551724138:

--------------------------------------------------------------------------

---------------------------------------------------------------------------

Top 10 Coefficients (based on absolute value) with sign

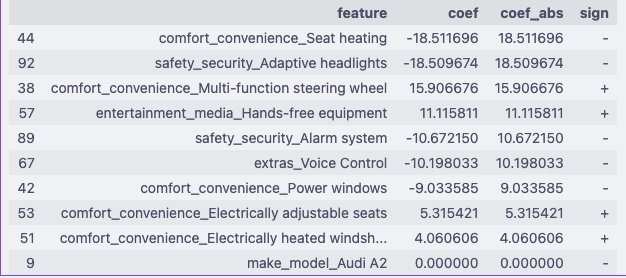
---------------------------------------------------------------------------



---------------------------------------------------------------------------

Bottom 10 Coefficients (based on absolute value) with sign

---------------------------------------------------------------------------



### Evaluate the Model on Test Data

* Display the values from test data model performance

Final Test Evaluation:

R²: 0.8951

MAE: 1323.56

MSE: 3715991.07

## Lasso Regression Implementation

### Define Alpha Range

We will start with alpha range of [ 0.01, 0.1, 1, 10, 50, 100]

### Apply Lasso Regularization, Find Chosen Alpha, Analyze Model Performance

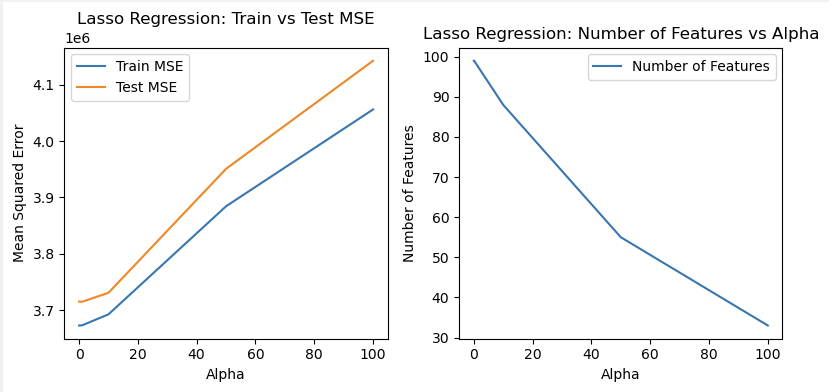
#### Apply Lasso Regularization

Fits Lasso models for a range of alpha values and returns train and test MSE scores along with the number of features used (non-zero coefficients) for each alpha. We will compare this with the best alpha coming from LassoCV. Since LassoCV uses MSE by default for hypertuning, we will use the same here.

* For each value of alpha in the range
  + Initialize Lasso() model
  + Fit the model on training data
  + Predict the y target on training and test data both
* Find the train and test scores using mean\_squarred\_error

#### Plot Error-Alpha chart

* Plots the training and testing MAE scores against alpha values for Lasso regression.
* Also plots the number of features used against alpha values.



#### Find the Chosen Alpha

* Select the alpha such that the test scores are minimum

We use min Test MSE, to compare with LassoCV later which by default uses MSE

Chosen alpha (Lasso): 1

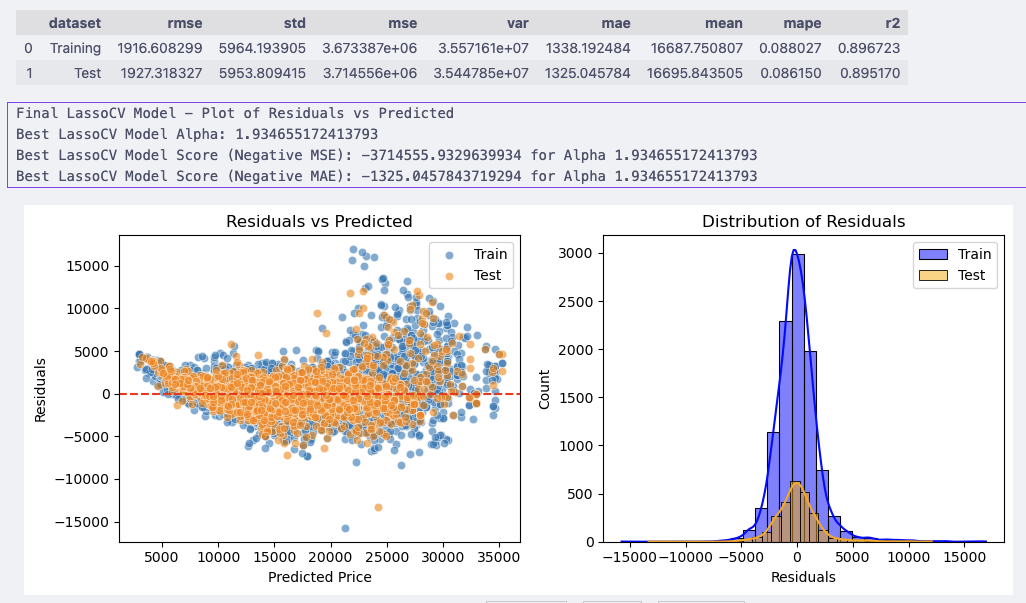
Chosen alpha (Lasso) with minimum Test MSE: 1, Test MSE: 3714657.239348934

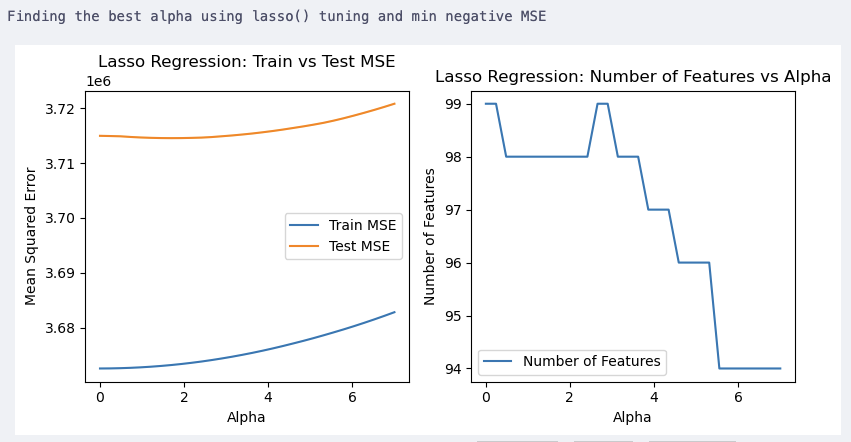
### Fine Tune with Narrow Alpha Range

* We use narrow alpha range : np.linspace(0.005, 7, 30) # 30 values between 0.005 and 10
* Fit a LassoCV model to the training data and returns predictions for both train and test sets. Also returns the chosen alpha value
* Initialize LassoCV with narrow alpha range and *cv*=*KFold*(*n\_splits*=5, *shuffle*=True, *random\_state*=42)
  + There is no scoring parameter in LassoCV, it uses MSE by default
  + cv=5 is same as default KFold(n\_splits=5, shuffle=False)
  + cv=KFold(n\_splits=5, shuffle=True, random\_state=42)
    - shuffle=True randomizes the order of the data before splitting
    - random\_state=42 ensures reproducibility (the same splits every time you run)
* Fit the model on training data
* Predict y, for train and test datasets
* Find best alpha using lassocv.alpha\_
* Find best score MAE and MSE for the test predicted and test actual values
* Analyze model performance using the reusable function
* Plot the train and test scores for narrow alpha range using the reusable function

Chosen alpha (using LassoCV): 1.934655172413793

Final LassoCV Model - Analyze Model Performance





We use min Test MSE, to compare with LassoCV later which by default uses MSE

Chosen alpha (Lasso): **1.69**34482758620688

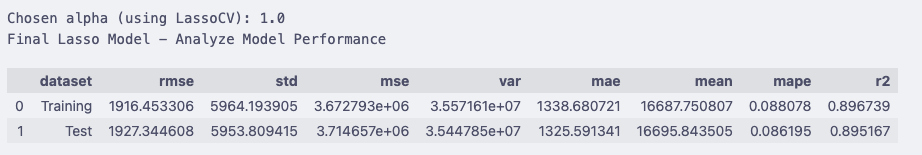
Chosen alpha (Lasso) 1.6934482758620688 with minimum Test MSE: **3714545**.352458124

Inference:

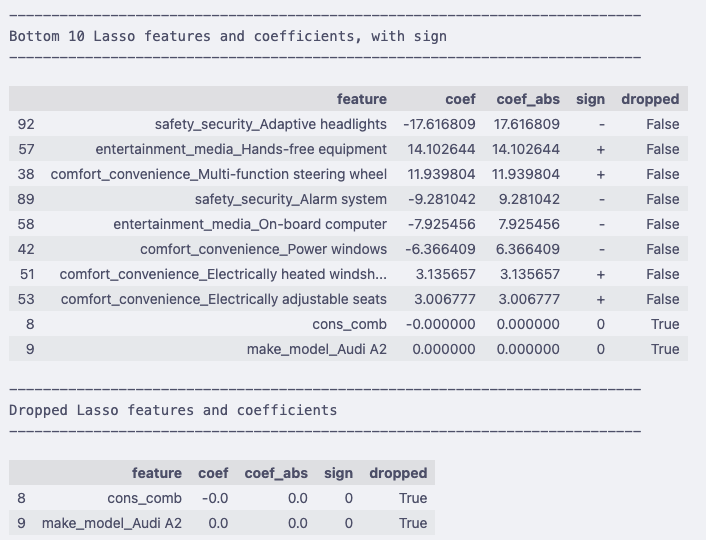
* It is very common to see different best α between LassoCV and your manual lasso() loop.
* Why the difference happens
  + Search space resolution
    - LassoCV does an internal search over the alphas you give it.
    - Manual loop only checks the discrete values you list.
    - Example: If the true optimal alpha is 0.015, and you only test [0.01, 0.1, 1], your loop might pick 0.01. LassoCV might interpolate and find 0.015 or return something close like 1.42 if its internal scoring favors it.
  + Cross-validation strategy
    - LassoCV uses its own internal CV splits (controlled by cv param, but defaults can differ).
    - Manual loop uses cross\_val\_score with a specific CV splitter (KFold, shuffle, random\_state) or calculates values manually without CV split.
    - Even tiny differences in how folds are split can shift the chosen alpha.
  + Randomness in splits / solver
    - Even with the same folds, Lasso is iterative and uses coordinate descent. If random\_state isn’t fixed in both, results can drift slightly.
  + Path vs fixed alphas
    - LassoCV traces along a regularization path (solution path for many alphas), and may “settle” on a point that minimizes CV-MSE along that path.
    - Manual loop just brute-forces and picks the discrete minimum.

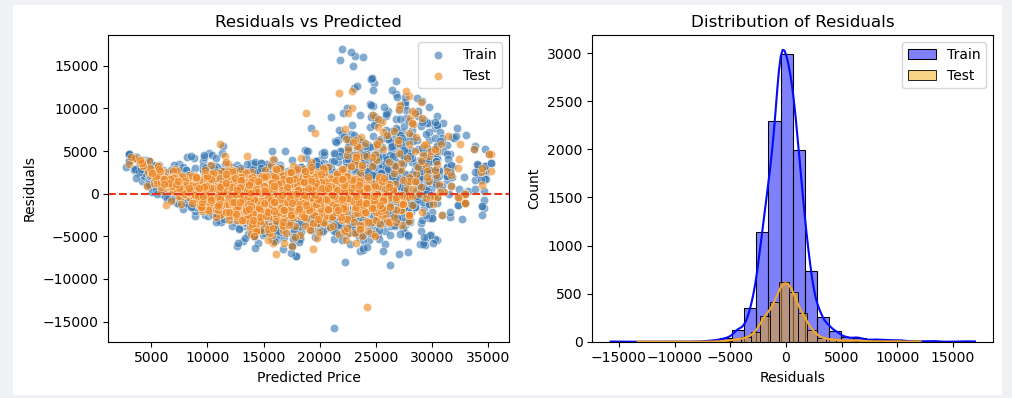
#### Run the Model using Best Alpha

* Best Alpha = 1.69
* Use reusable function of lassocv to fit the model on training data and evaluate on test data
* Analyze model performance
* Get the coefficients of the fitted model
* Print the top-n coefficients by absolute value









#### Evaluate Model on Test Data

* Use the reusable function to show the scores on test data

Final Test Evaluation:

R²: 0.8952

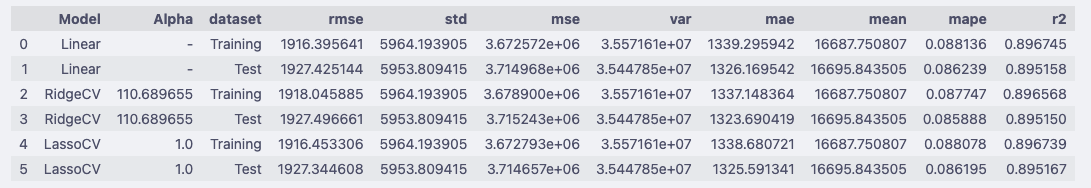
MAE: 1325.05

MSE: 3714555.93

## Regularization Comparison & Analysis

### Compare Evaluation Metrics of Each Model

* Get the 3 output datasets from analyze model performance function for the respective LR, RIDGECV and LASSOCV models
* Add 2 columns with hold the name of the model and respective best alpha scores
* Combine the datasets and display the metrics



#### Interpretation strategy

* If Linear Regression has the lowest test error → data is not very multicollinear, and regularization doesn’t add much.
* If Ridge improves test performance slightly but reduces variance → multicollinearity was an issue, Ridge is stabilizing coefficients.
* If Lasso performs better (or gives similar error with fewer features) → many coefficients were not important, and Lasso added interpretability by feature selection.
* If all three are almost the same → dataset may be “well-behaved,” and regularization isn’t critical here.

### Compare Coefficients of the 3 models

* Use the model\_coeff datasets created in the functions for the respective 3 models
* Plot bar chart with top-10 features based on absolute value of the coefficients for all 3 models
* Plot bar chart with bottom-10 features based on absolute value of the coefficients for all 3 models
* For LassoCV:
  + Plot bar chart with top-10 features based on absolute value of the coefficients
  + Plot bar chart with bottom-10 features based on absolute value of the coefficients
  + Plot bar chart with eliminated features based on zero value of the coefficients

A graph with different colored bars

AI-generated content may be incorrect.

A graph with different colored bars

AI-generated content may be incorrect.

A graph with different colored bars

AI-generated content may be incorrect.

# Conclusion

## Model Performance Comparison Conclusion

### Observations

* All three models perform almost identically
  + **RMSE ≈ 1927**
  + **MAE ≈ 1325**
  + **R² ≈ 0.895**

→ Predictive accuracy is effectively the same.

* RidgeCV
  + Chose **α ≈ 110.7**
  + Reduced MAE slightly compared to baseline (**1323 vs 1326**).
  + Regularization didn’t change predictive performance much, but it **stabilizes coefficients** (reduces multicollinearity risk).
* LassoCV
  + Chose **α = 1.0**
  + Similar performance to baseline and Ridge.
  + Main benefit: **feature selection** (shrinks/zeroes out less important coefficients).
* Baseline Linear Regression
  + Already very strong (\*\***R² ≈ 0.895**\*\*).
  + Regularization doesn’t improve error metrics, but might affect interpretability.

### Summary

* **Predictive power**: All three models are essentially equivalent in performance (differences are in the 3rd decimal place).
* **RidgeCV**: Best if your goal is to handle multicollinearity and keep all features with shrunk coefficients.
* **LassoCV**: Best if you want to simplify the model by feature selection (dropping irrelevant predictors).
* **Baseline Linear Regression**: Already very good; regularization doesn’t add predictive accuracy but can help with coefficient stability (Ridge) or sparsity (Lasso).

### Predictors and Coefficients Comparison

#### Top Predictors across Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Linear Regression | Ridge Regression | Lasso Regression | Interpretation |
| make\_model\_Opel Corsa | -1873.56 | -1767.95 | -1869.27 | Strong negative effect; very stable across models. |
| hp\_kw | +1786.34 | +1702.84 | +1777.46 | Strongest positive driver; Ridge shrinks slightly. |
| make\_model\_Renault Clio | -1730.77 | -1609.58 | -1720.82 | Consistently strong negative effect. |
| make\_model\_Opel Astra | -1268.10 | -1155.89 | -1259.25 | Stable negative contribution. |
| km | -1215.94 | -1204.06 | -1214.45 | Mileage reduces price; very consistent. |
| age | -1195.50 | -1191.24 | -1198.52 | Older cars worth less; unchanged across models. |
| gearing\_type\_Manual | -807.50 | -799.54 | -809.74 | Manual gearboxes lower price; effect stable. |
| make\_model\_Renault Espace | +784.07 | +804.01 | +786.73 | Model-specific premium; Ridge inflates slightly. |
| type\_Used | -710.31 | -709.05 | -708.49 | Used cars discounted; identical across models. |
| make\_model\_Audi A3 | +676.51 | +701.38 | +677.35 | Positive Audi brand premium; stable. |

### Bottom Predictors across Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Linear Regression | Ridge Regression | Lasso Regression | Interpretation |
| previous\_owners | -18.57 | – | – | Weak effect, not picked up in Ridge/Lasso. |
| Hands-free equipment | +18.03 | +11.12 | +14.10 | Very small premium for feature. |
| Multi-function steering wheel | +13.83 | +15.91 | +11.94 | Consistently tiny effect across models. |
| Alarm system | -11.41 | -10.67 | -9.28 | Slight negative effect, unexpected. |
| On-board computer | -8.43 | – | -7.93 | Negligible impact, shrunk/dropped. |
| Power windows | -7.81 | -9.03 | -6.37 | Effectively irrelevant, but not dropped. |
| Electrically adjustable seats | +4.58 | +5.32 | +3.01 | Very small positive contribution. |
| cons\_comb | +4.47 | – | 0.00 (dropped) | Lasso eliminates it, shows redundancy with hp/displacement. |
| Heated windscreen | +4.35 | +4.06 | +3.14 | Very minor effect. |
| make\_model\_Audi A2 | ~0.00 | 0.00 | 0.00 (dropped) | Irrelevant; Lasso drops completely. |
| Seat heating | – | -18.51 | – | Ridge detects weak negative effect, others ignore. |
| Adaptive headlights | – | -18.51 | -17.62 | Tiny negative, only kept in Ridge/Lasso. |
| Voice Control | – | -10.20 | – | Negligible; shrunk by Ridge, ignored elsewhere. |

#### Consistency in Top Drivers

Across all three models, the same top features dominate price prediction:

* **hp\_kw, km, age, Opel/Renault model dummies, gear type, car type**
* Signs and magnitudes are stable (Ridge shrinks them slightly).

#### Regularization Impact

* **Ridge:** Shrinks extreme coefficients (Opel Corsa, Renault Clio, hp\_kw) but keeps all features.
* **Lasso:** Drops weak/irrelevant features (e.g., Audi A2, cons\_comb → coefficients forced to 0).

#### Bottom Features Behavior

* **Linear Regression:** Leaves tiny but noisy coefficients (e.g., cons\_comb small but nonzero).
* **Ridge:** Shrinks these to small values but never to zero.
* **Lasso:** Removes them entirely → cleaner model, better interpretability.

#### Summary

* **If interpretability and feature selection matter → Lasso wins.**
* **If keeping all features while handling collinearity → Ridge is safer.**
* **If you just want predictive accuracy → even baseline Linear is already strong.**

#### Business Takeaway

* **Key Price Drivers Are Clear and Stable**
  + Engine power (**hp\_kw**), mileage (**km**), car age, and certain premium/budget models (Opel/Renault/Audi dummies) consistently dominate price prediction across all models.
  + This confirms where pricing focus should be: **high-power, newer, low-mileage, premium models command higher prices**, while older, used, and manual vehicles depress resale value.
* **Regularization Supports Decision-Making**
  + **Ridge regression** stabilizes coefficients, reducing risks of overestimating correlated features like engine size and weight.
  + **Lasso regression** identifies irrelevant or redundant features (e.g., cons\_comb, Audi A2) that do not meaningfully influence pricing, simplifying analysis and reporting.
* **Actionable Insight for Pricing Strategy**
  + Focus marketing and sales efforts on features that consistently boost value (premium models, high engine power, low mileage).
  + Avoid overcomplicating pricing models with low-impact features; Lasso highlights which features can be safely ignored.
  + Use Ridge when all features matter but multicollinearity exists, ensuring stable coefficient estimates for decision-making.
* **Predictive Accuracy vs Interpretability**
  + All three models (Linear, Ridge, Lasso) predict price with near-identical accuracy (R² ≈ 0.895).
  + **Decision-makers should choose model type based on business need:**
  + Simplified, interpretable pricing → **Lasso**
  + Robust estimates including all features → **Ridge**
  + Quick baseline insights → **Linear regression**