1 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 regularization 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.
- **Feature Importance**: Identify the most important factors influencing car prices

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.

1.1 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

1.1.1 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)
```

2 Analysis and Feature Engineering

2.1 Preliminary Analysis and Frequency Distributions

2.1.1 Handle Missing Values

2.1.1.1 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',	Mode	Discrete few values

'safety_security', 'paint_type', 'upholstery_type', 'gearing_type', 'drive_chain'		

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

2.1.1.2 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

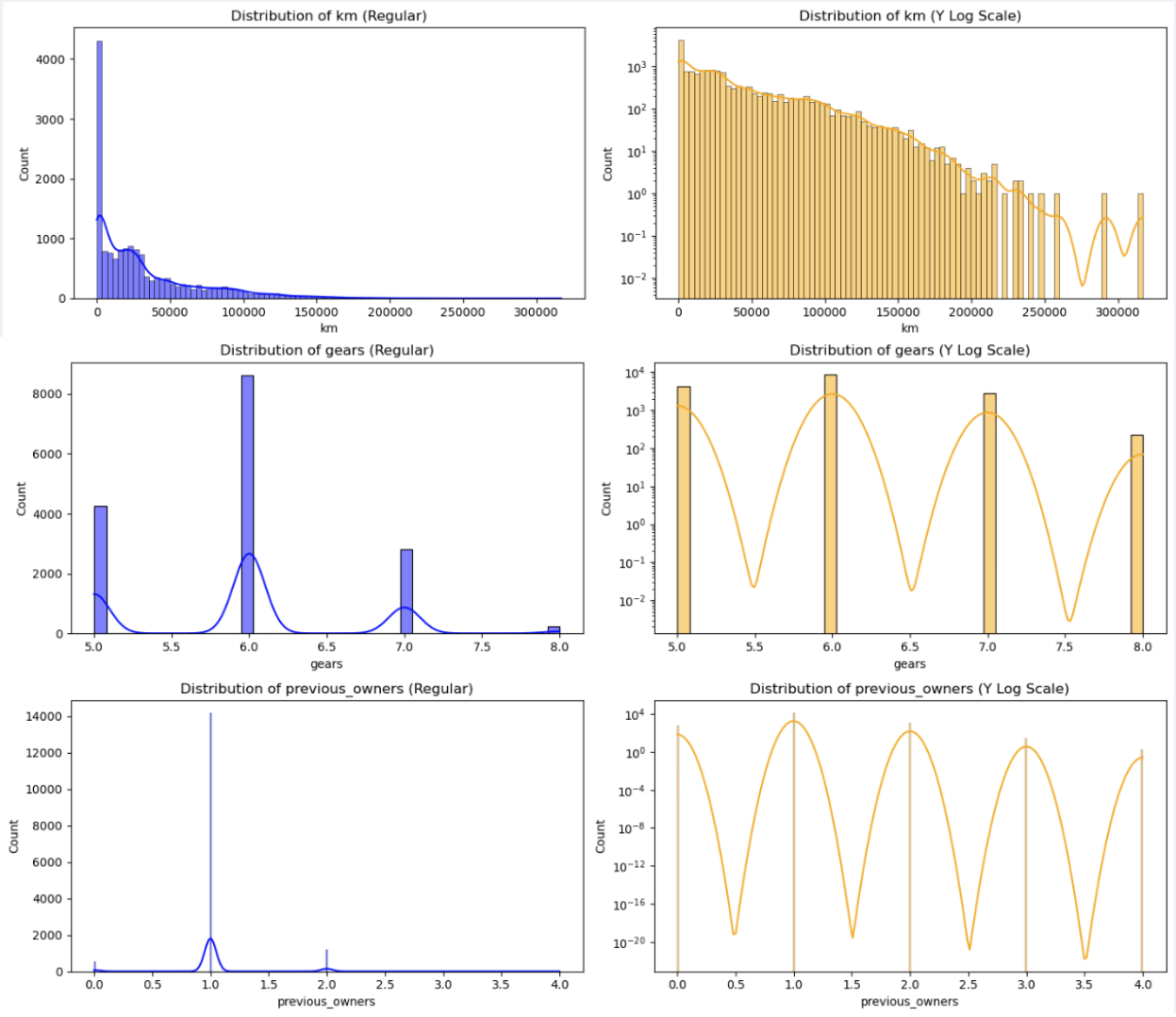
Results:

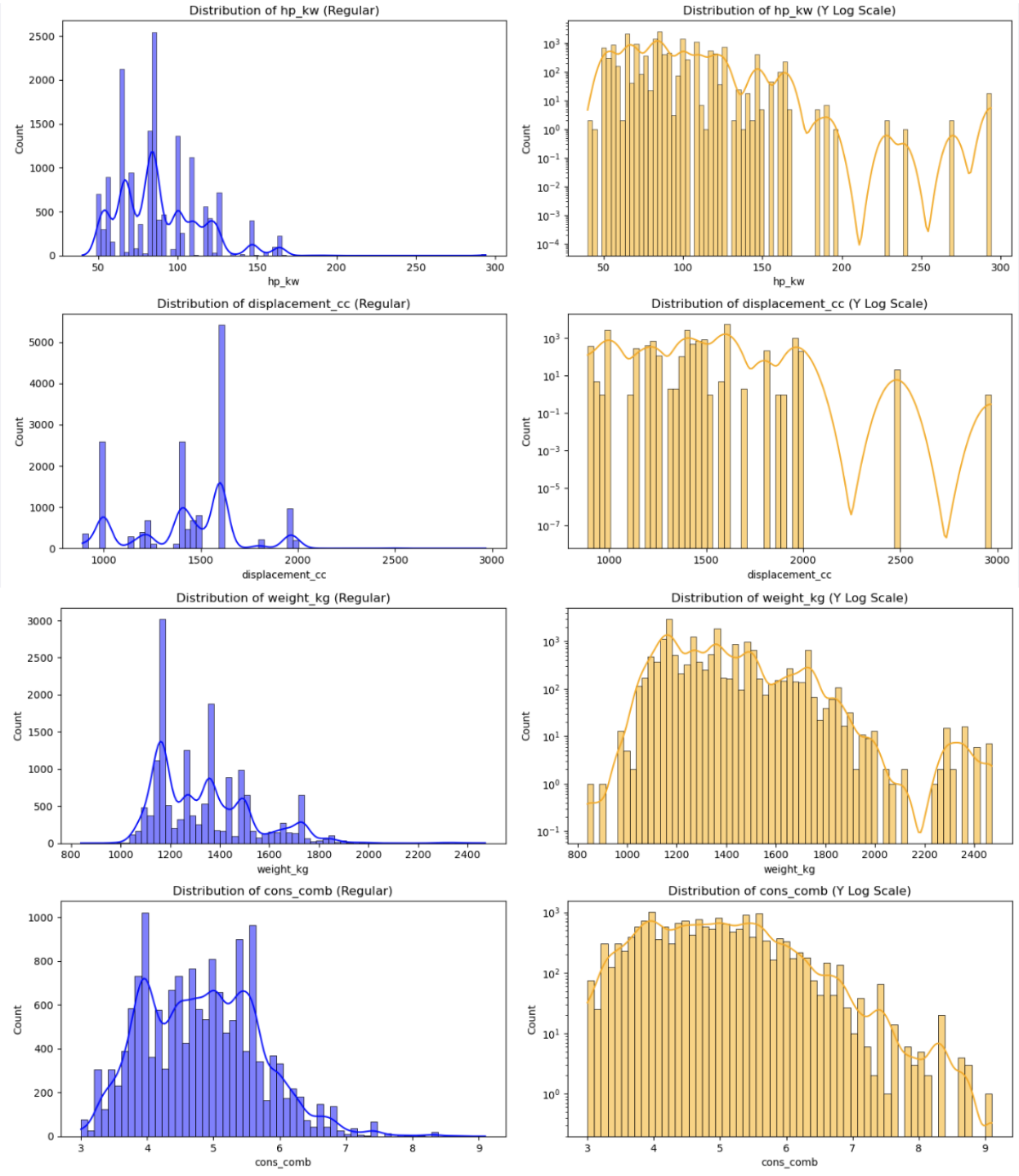
Column Name	Data Type	Missing Count	Missing %	Impute Type	Action
make_model	object	0	0.0	mode	none
age	float64	0	0.0	median	none
drive_chain	object	0	0.0	mode	none
weight_kg	float64	0	0.0	mean	none
displacement_cc	float64	0	0.0	mean	none
gearing_type	object	0	0.0	mode	none
upholstery_type	object	0	0.0	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

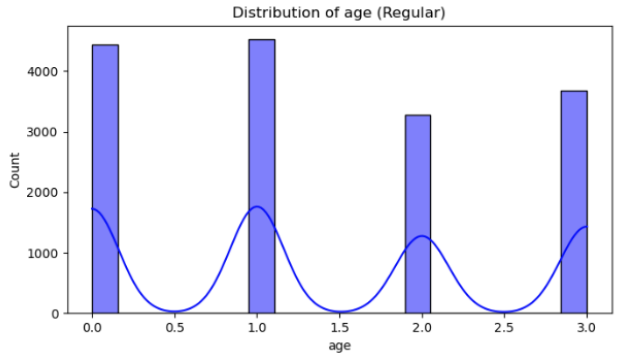
entertainment_media	object	0	0.0	mode	none
comfort_convenience	object	0	0.0	mode	none
gears	float64	0	0.0	mode	none
fuel	object	0	0.0	mode	none
type	object	0	0.0	mode	none
km	float64	0	0.0	median	none
vat	object	0	0.0	mode	none
price	int64	0	0.0	drop	none
cons_comb	float64	0	0.0	mean	none

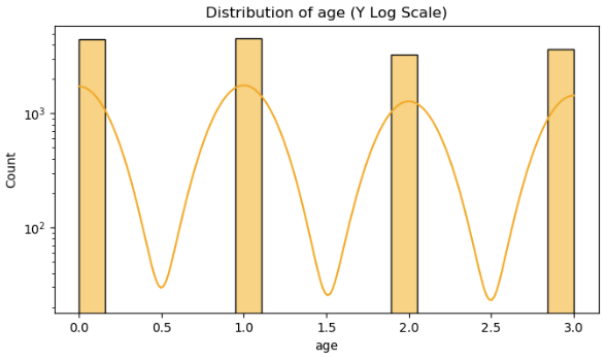
2.1.2 Identify Numerical Predictors and Plot Frequency Distribution

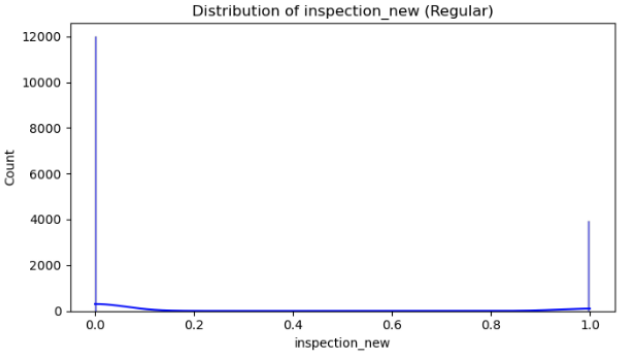
- Create 2 histogram plots per numeric column
  - First plot using regular yscale
  - Second plot using yscale = 'log'
- Plot box plot and violin plot for the numeric columns

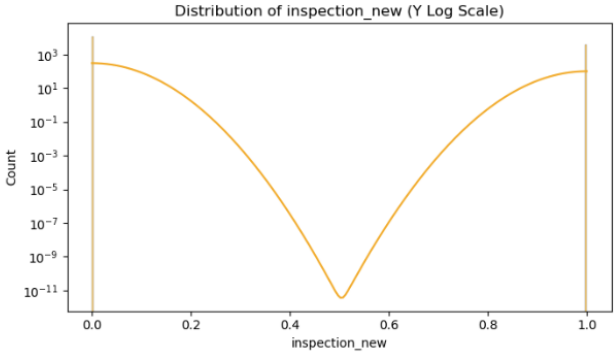


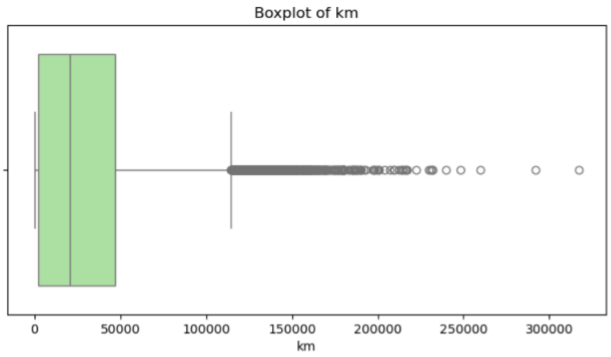

Distribution of hp_kw (Regular)
Count
hp_kw
Distribution of hp_kw (Y Log Scale)
Count
hp_kw
Distribution of displacement_cc (Regular)
Count
displacement_cc
Distribution of displacement_cc (Y Log Scale)
Count
displacement_cc
Distribution of weight_kg (Regular)
Count
weight_kg
Distribution of weight_kg (Y Log Scale)
Count
weight_kg
Distribution of cons_comb (Regular)
Count
cons_comb
Distribution of cons_comb (Y Log Scale)
Count
cons_comb

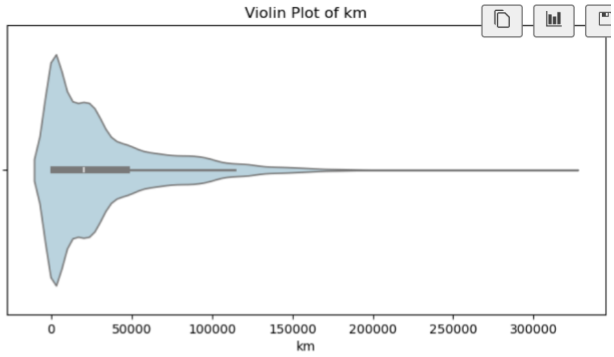

Distribution of age (Regular)
Count
age

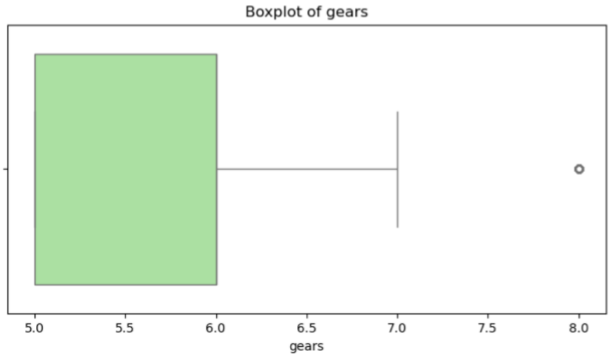

Distribution of age (Y Log Scale)
Count
age

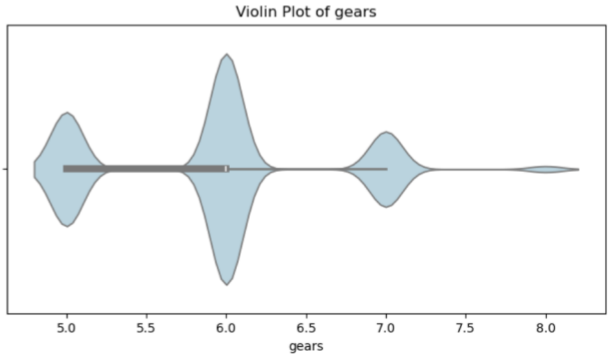

Distribution of inspection_new (Regular)
Count
inspection_new

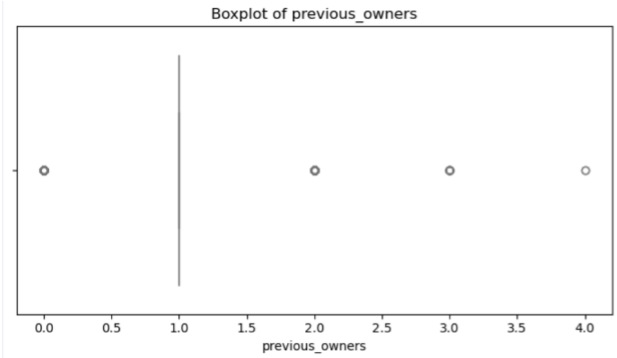

Distribution of inspection_new (Y Log Scale)
Count
inspection_new

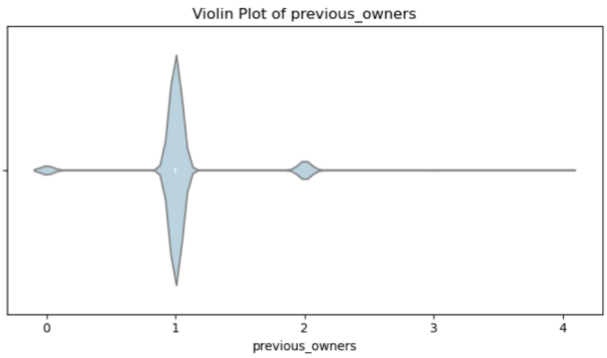

Boxplot of km
km

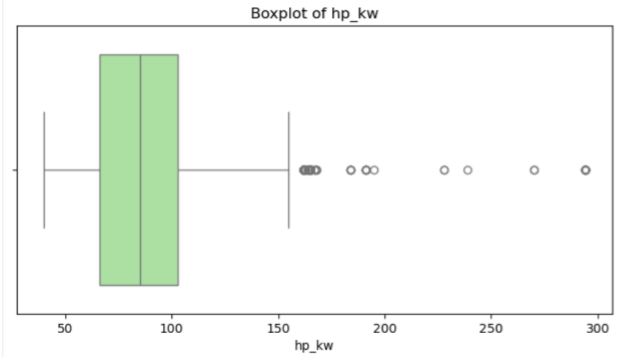

Violin Plot of km
km

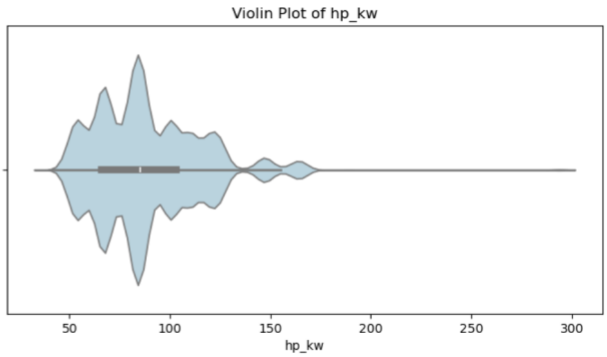

Boxplot of gears
gears

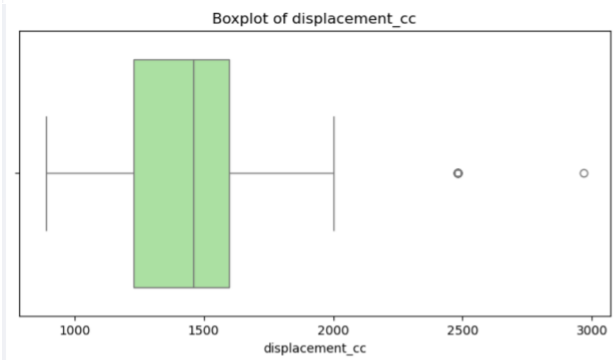

Violin Plot of gears
gears

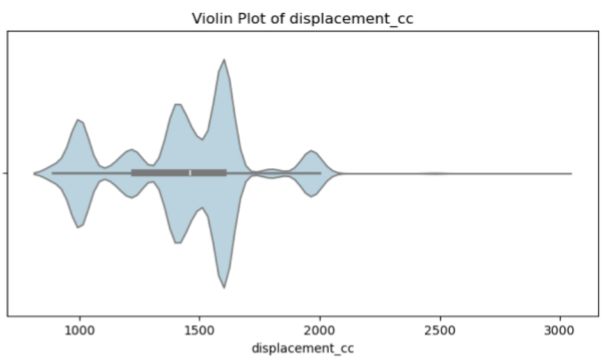

Boxplot of previous_owners
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
previous_owners

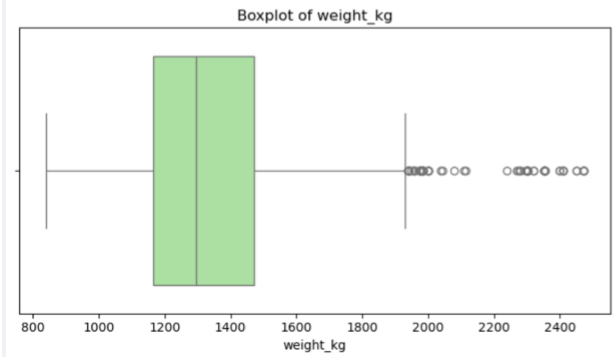

Violin Plot of previous_owners
0
1
2
3
4
previous_owners


Boxplot of hp_kw
50
100
150
200
250
300
hp_kw

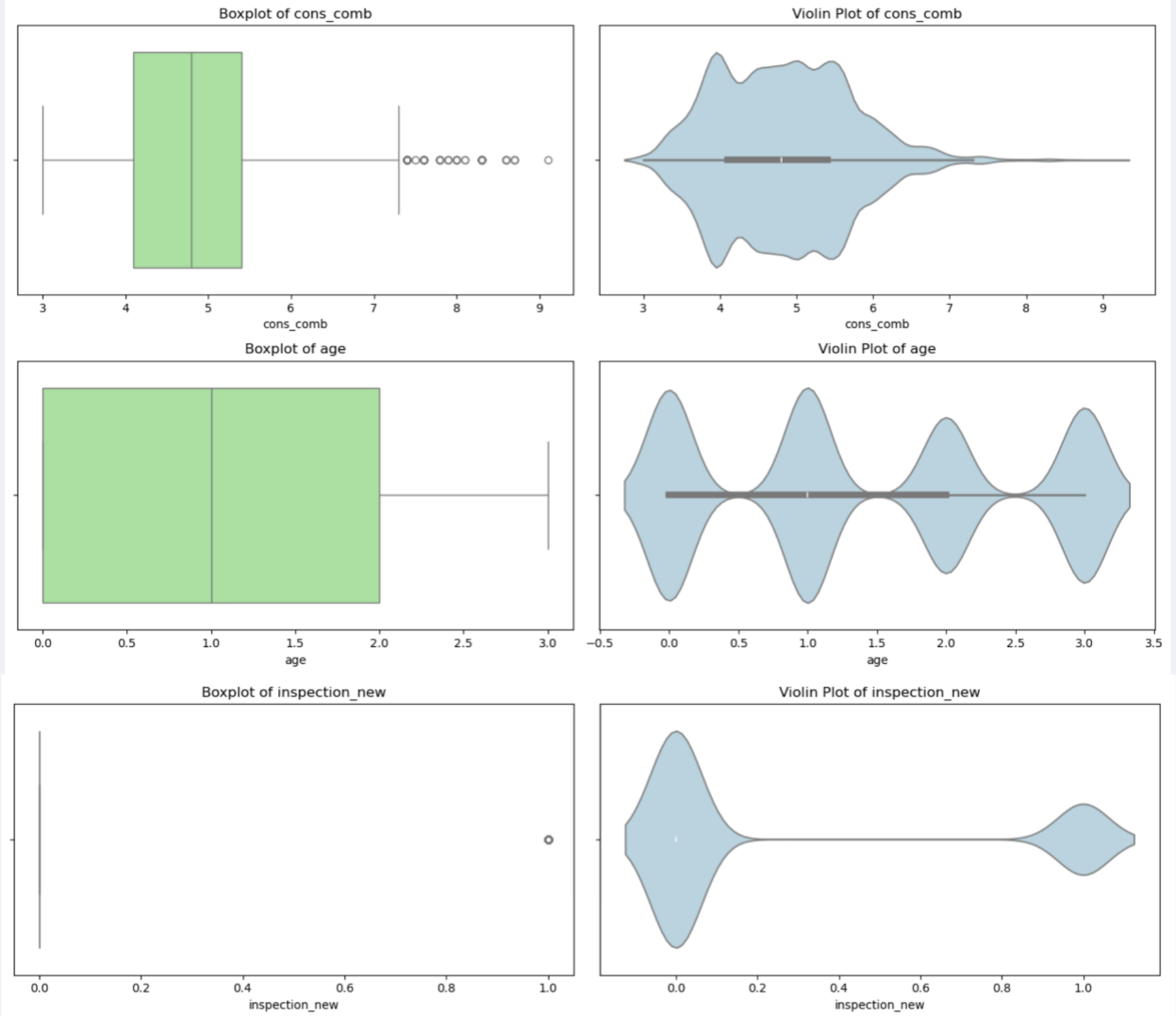

Violin Plot of hp_kw
50
100
150
200
250
300
hp_kw


Boxplot of displacement_cc
1000
1500
2000
2500
3000
displacement_cc


Violin Plot of displacement_cc
1000
1500
2000
2500
3000
displacement_cc

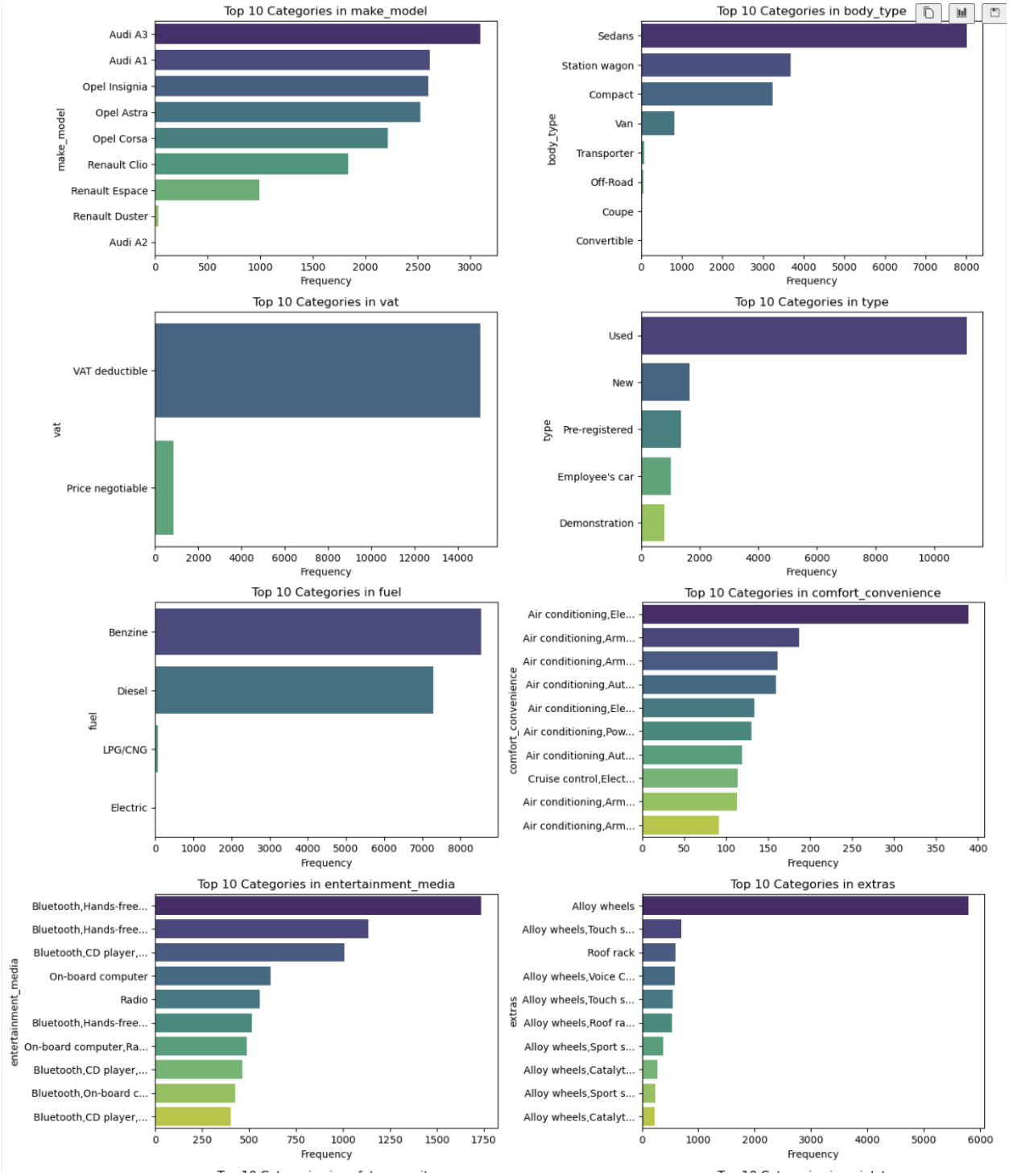

Boxplot of weight_kg
800
1000
1200
1400
1600
1800
2000
2200
2400
weight_kg

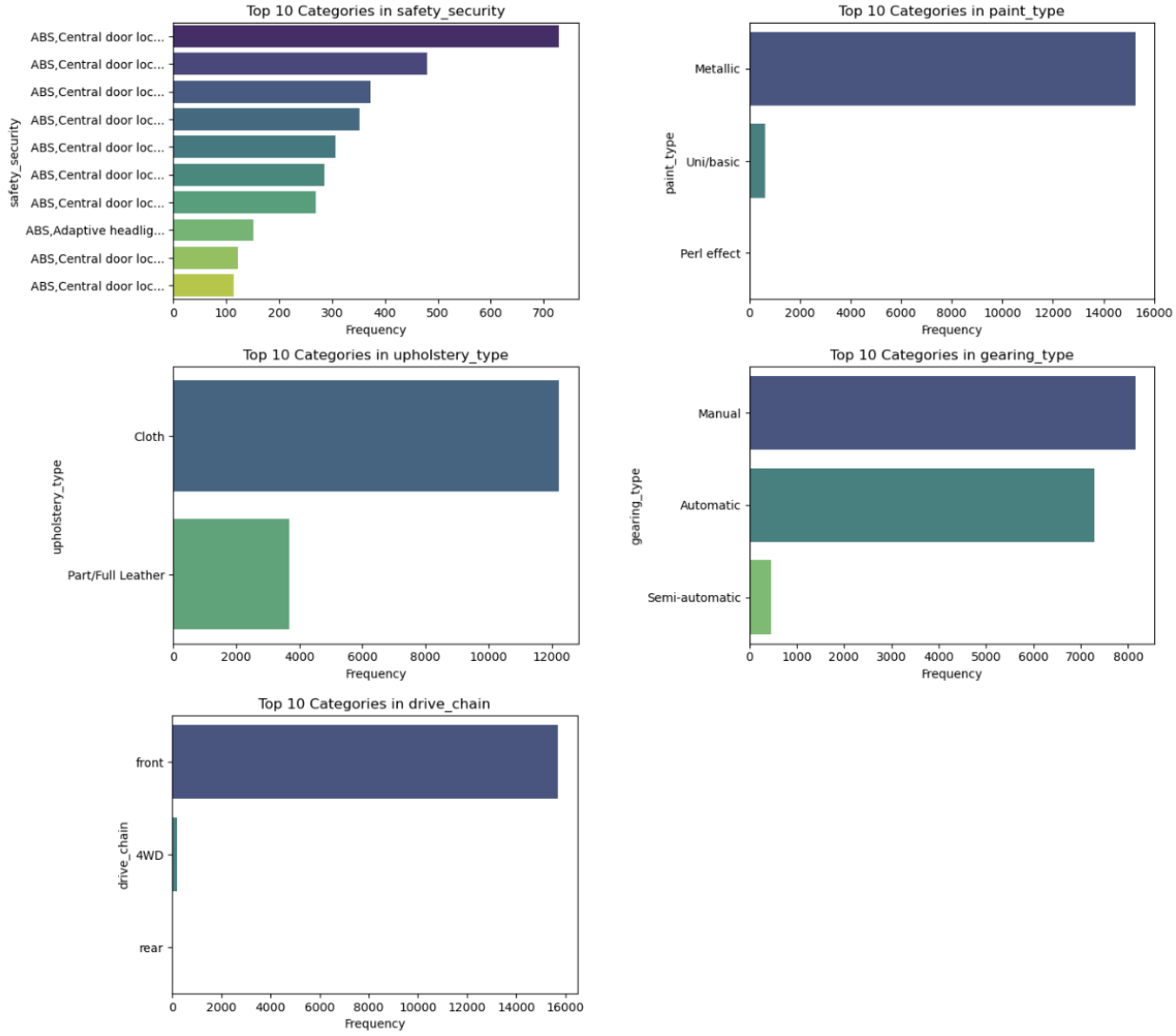

Violin Plot of weight_kg
750
1000
1250
1500
1750
2000
2250
2500
weight_kg



2.1.3 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


Top 10 Categories in make_model
Audi A3
Audi A1
Opel Insignia
Opel Astra
Opel Corsa
Renault Clio
Renault Espace
Renault Duster
Audi A2
make_model
0 500 1000 1500 2000 2500 3000
Frequency
Top 10 Categories in body_type
Sedans
Station wagon
Compact
Van
Transporter
Off-Road
Coupe
Convertible
body_type
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency
Top 10 Categories in vat
VAT deductible
Price negotiable
vat
0 2000 4000 6000 8000 10000 12000 14000
Frequency
Top 10 Categories in type
Used
New
Pre-registered
Employee's car
Demonstration
type
0 2000 4000 6000 8000 10000
Frequency
Top 10 Categories in fuel
Benzine
Diesel
LPG/CNG
Electric
fuel
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency
Top 10 Categories in comfort_convenience
Air conditioning,Ele...
Air conditioning,Arm...
Air conditioning,Arm...
Air conditioning,Aut...
Air conditioning,Ele...
Air conditioning,Pow...
Air conditioning,Aut...
Cruise control,Elect...
Air conditioning,Arm...
Air conditioning,Arm...
comfort_convenience
0 50 100 150 200 250 300 350 400
Frequency
Top 10 Categories in entertainment_media
Bluetooth,Hands-free...
Bluetooth,Hands-free...
Bluetooth,CD player,...
On-board computer
Radio
Bluetooth,Hands-free...
On-board computer,Ra...
Bluetooth,CD player,...
Bluetooth,On-board c...
Bluetooth,CD player,...
entertainment_media
0 250 500 750 1000 1250 1500 1750
Frequency
Top 10 Categories in extras
Alloy wheels
Alloy wheels,Touch s...
Roof rack
Alloy wheels,Voice C...
Alloy wheels,Touch s...
Alloy wheels,Roof ra...
Alloy wheels,Sport s...
Alloy wheels,Catalyt...
Alloy wheels,Sport s...
Alloy wheels,Catalyt...
extras
0 1000 2000 3000 4000 5000 6000
Frequency



- 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.

2.1.4 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

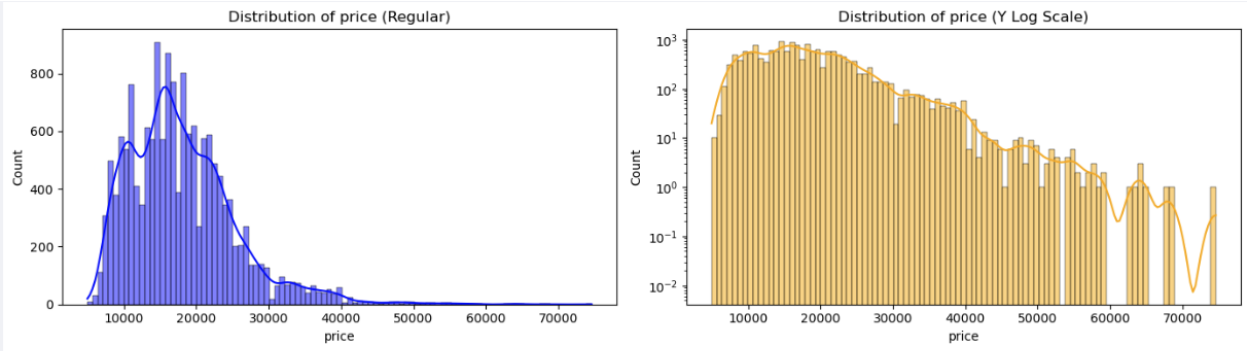
```
--------------------------------------------------------------------------------
Value counts for column 'type':
type
Used              0.697141
New               0.103613
Pre-registered    0.085705
Employee's car    0.063525
Demonstration     0.050016
Name: proportion, dtype: float64

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
--------------------------------------------------------------------------------
```

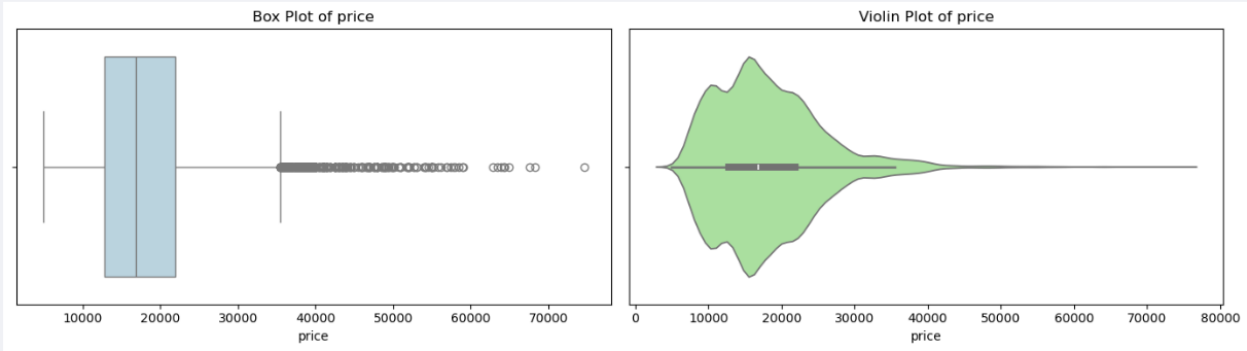
```
Value counts for column 'fuel':
fuel
Benzine     0.537103
Diesel      0.458561
LPG/CNG     0.004021
Electric    0.000314
Name: proportion, dtype: float64

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
```

2.1.5 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 transformation
    - Check original skewness
    - Check there are no negative or zero values
    - Apply lop1p transformation on Y
    - Check skewness after applying log transformation
    - If absolute value of post-transformation skew
        - <0.5 – fairly symmetric
        - 0.5 to 1 – moderately skewed
        - >1 – highly skewed
- Plot histogram, box plot, violin plot on 'price' post outlier treatment



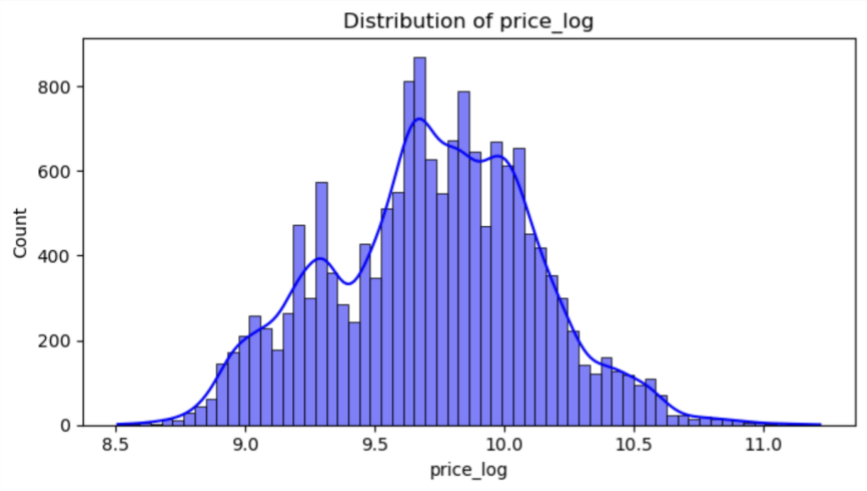


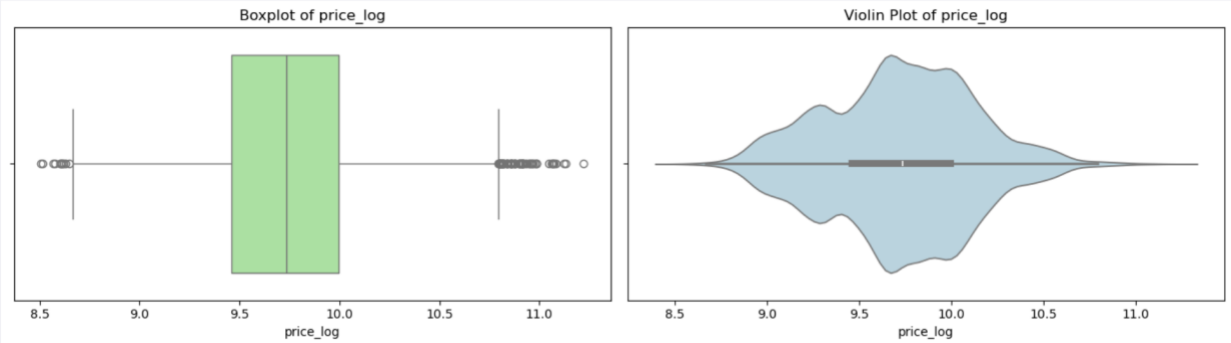
```
Applying log transformation to target column 'price' to reduce skewness.
Original skewness of 'price': 1.236169412899669
negatives: 0, zeros: 0
After log1p: skew=-0.031
Log transformation output : fairly symmetric (|skew| < 0.5)
```

Skewness value	Interpretation	Visual shape	Meaning for regression
0	Perfectly symmetric	Bell-shaped	Ideal for linear regression (normal residuals).
> 0 (positive skew)	Right tail longer (many small values, few large ones)	Right-heavy	Common for prices, incomes, sales — most are small, few are huge.
< 0 (negative skew)	Left tail longer (many large values, few small ones)	Left-heavy	Rare in price data; usually means a floor effect.

```
Check few outlier values for high end cars
Mean price for Audi A3: 20996.693251533743
Mean price for Renault Espace: 30080.21190716448
```

```
Post Log Transformation Treatment on ['price_log']:
```





2.2 Correlation Analysis

Use the transformed target (price_log) for correlation analysis

Why:

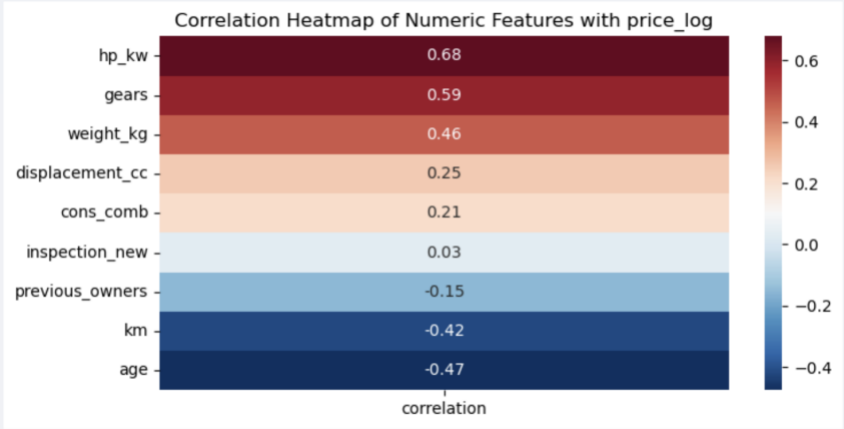
- We're building your model on the transformed target, so all pre-model analyses (correlation, feature selection, etc.) should reflect relationships with that version of the target.
- **Log transformation linearizes nonlinear relationships**.

- Many numerical predictors (like hp_kw, km, age) have multiplicative or exponential effects on price.
- Taking log(price) converts those into roughly additive linear relationships — exactly what correlation (and linear regression) assume.

- **Pearson correlation assumes linearity and roughly normal variables.**
  - The original price was right-skewed (skew ≈ +1.24), violating this assumption.
  - price_log (skew ≈ 0) satisfies it much better, making correlations meaningful and less dominated by a few luxury-car outliers.
- **Mutual information for categorical features**: is nonparametric and scale-invariant, so it's fine either way — but to keep consistency, use the same target (price_log) everywhere.

2.2.1 Correlation Map Between Features and Target Variable

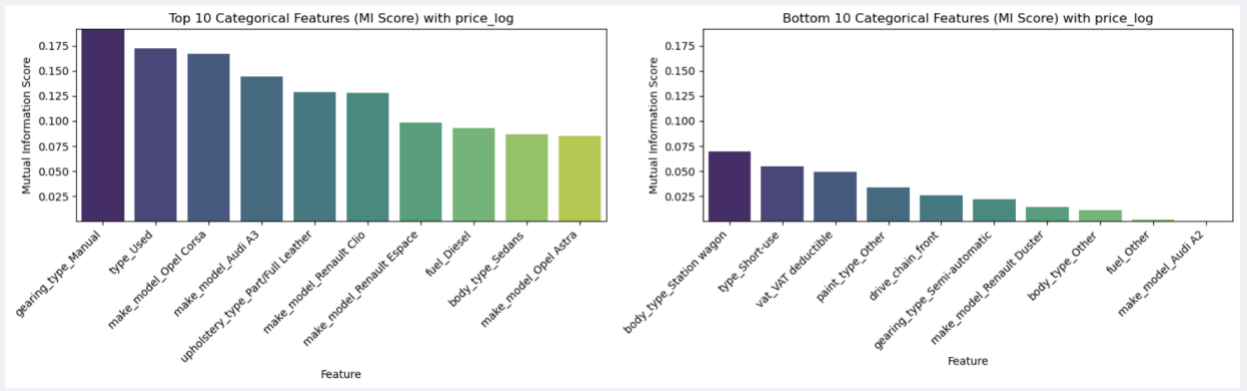
- Separate features and target into X and y
- Find correlation of numeric columns with target variable 'price_log'
- Create a data frame with feature name, data type, abs(score), sign
- Plot a heatmap of numeric features correlation
- Display the numeric features with positive or negative high correlation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



	feature	type	score	sign
0	hp_kw	numeric	0.678024	+
1	gears	numeric	0.588194	+
2	age	numeric	0.474618	-
3	weight_kg	numeric	0.464597	+
4	km	numeric	0.419189	-
5	displacement_cc	numeric	0.254854	+
6	cons_comb	numeric	0.211097	+
7	previous_owners	numeric	0.152029	-
8	inspection_new	numeric	0.030927	+

2.2.2 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 column price_log
- 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



MI Score Ranking of Top 10 Categorical Features with price_log:

	feature	type	score	sign
0	gearing_type_Manual	categorical	0.191521	None
1	type_Used	categorical	0.172049	None
2	make_model_Opel Corsa	categorical	0.166655	None
3	make_model_Audi A3	categorical	0.144532	None
4	upholstery_type_Part/Full Leather	categorical	0.128890	None
5	make_model_Renault Clio	categorical	0.127724	None
6	make_model_Renault Espace	categorical	0.098145	None
7	fuel_Diesel	categorical	0.093154	None
8	body_type_Sedans	categorical	0.086926	None
9	make_model_Opel Astra	categorical	0.085342	None

MI Score Ranking of Bottom 10 Categorical Features with price_log:

	feature	type	score	sign
12	body_type_Station wagon	categorical	0.069255	None
13	type_Short-use	categorical	0.054666	None
14	vat_VAT deductible	categorical	0.049533	None
15	paint_type_Other	categorical	0.034203	None
16	drive_chain_front	categorical	0.025958	None
17	gearing_type_Semi-automatic	categorical	0.022270	None
18	make_model_Renault Duster	categorical	0.014348	None
19	body_type_Other	categorical	0.011160	None
20	fuel_Other	categorical	0.002250	None
21	make_model_Audi A2	categorical	0.000181	None

Combine the numeric and categorical features, and display the top-20 features with high correlation to target 'price_log'.

	feature	type	score	sign
0	hp_kw	numeric	0.678024	+
1	gears	numeric	0.588194	+
2	age	numeric	0.474618	-
3	weight_kg	numeric	0.464597	+
4	km	numeric	0.419189	-
5	displacement_cc	numeric	0.254854	+
6	cons_comb	numeric	0.211097	+
7	gearing_type_Manual	categorical	0.191521	None
8	type_Used	categorical	0.172049	None
9	make_model_Opel Corsa	categorical	0.166655	None
10	previous_owners	numeric	0.152029	-
11	make_model_Audi A3	categorical	0.144532	None
12	upholstery_type_Part/Full Leather	categorical	0.128890	None
13	make_model_Renault Clio	categorical	0.127724	None
14	make_model_Renault Espace	categorical	0.098145	None
15	fuel_Diesel	categorical	0.093154	None
16	body_type_Sedans	categorical	0.086926	None
17	make_model_Opel Astra	categorical	0.085342	None
18	body_type_Van	categorical	0.083550	None
19	make_model_Opel Insignia	categorical	0.077306	None

2.2.2.1 Feature Importance Summary

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

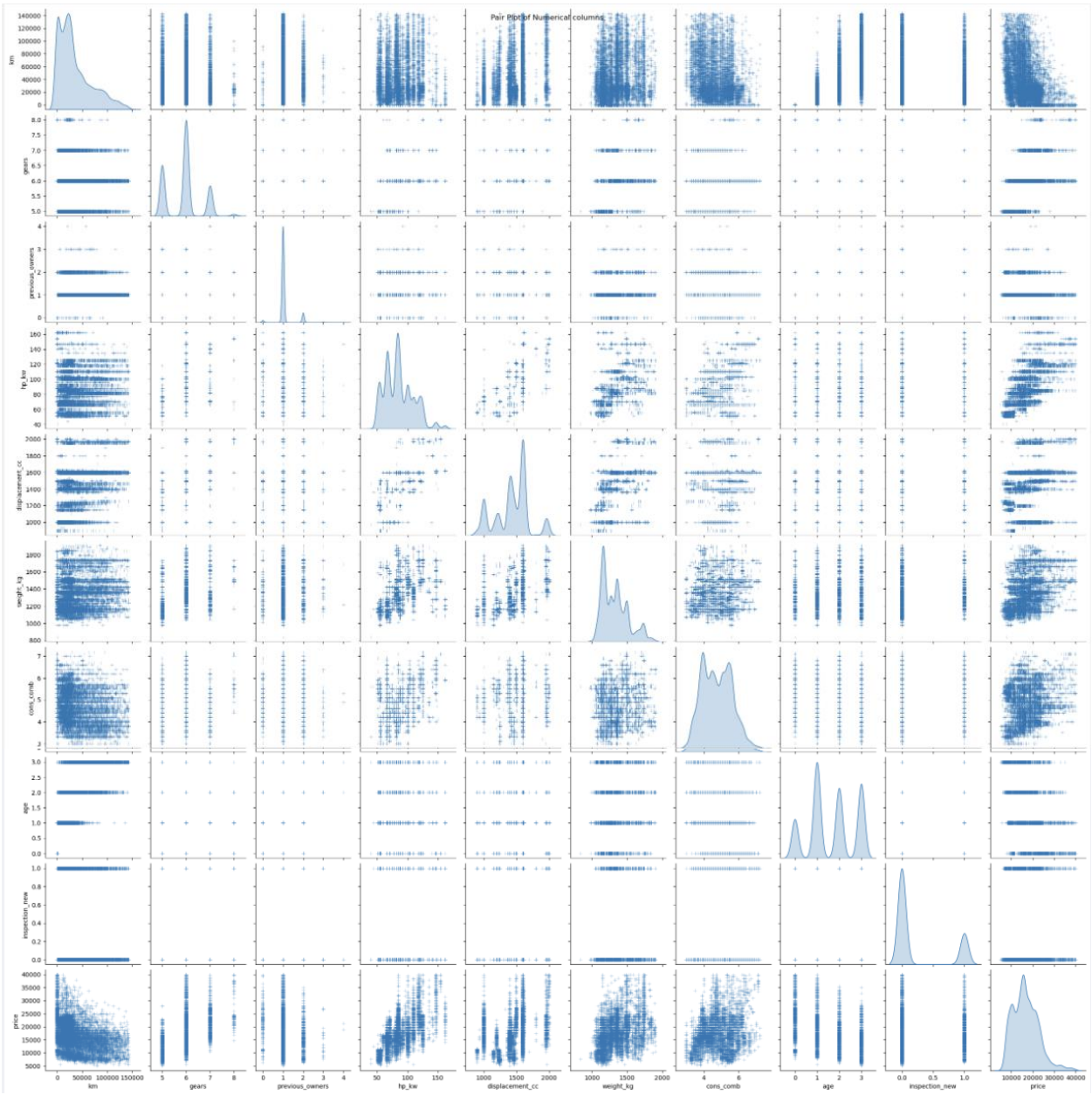
- **Numerical features** are ranked by absolute Pearson correlation with the target.
- **Categorical features** are ranked by mutual information (MI) with the target.
- This combined ranking helps prioritize which features to focus on for modeling and interpretation.

Rank	Feature	Type	Score	Sign	Inference
1	hp_kw	Numeric	0.668	+	Strongest driver of price. More engine power → higher price.
2	gears	Numeric	0.558	+	More gears (modern transmissions) → generally higher prices.
3	age	Numeric	0.475	–	Older cars lose value. Strong negative effect.
4	weight_kg	Numeric	0.465	+	Heavier cars (luxury/SUVs) are costlier.

5	km	Numeric	0.419	–	Higher mileage reduces price, though less than age.
6	displacement_cc	Numeric	0.225	+	Bigger engines → higher prices, though weaker than horsepower.
7	cons_comb	Numeric	0.211	+	Higher consumption → often premium/performance cars.
8	gearing_type_Manual	Categorical	0.192	–	Manual cars usually cheaper than automatics (signal of lower price).
9	type_Used	Categorical	0.172	–	Used cars priced lower than new ones.
10	make_model_Opel Corsa	Categorical	0.167	–	Budget model → pulls price down.
11	make_model_Audi A3	Categorical	0.145	+	Premium model → lifts price.
12	previous_owners	Numeric	0.152	–	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	0.098	–	Family van, not premium → lower prices.
16	fuel_Diesel	Categorical	0.093	±	Diesel cars affect price; historically valuable in EU, now mixed.
17	body_type_Sedans	Categorical	0.087	±	Sedans have moderate premium depending on market.
18	make_model_Opel Astra	Categorical	0.085	–	Mid-segment Opel → relatively lower priced.
19	make_model_Opel Insignia	Categorical	0.077	–	Slightly higher than Astra but still budget vs luxury brands.
20	body_type_Van	Categorical	0.084	–	Vans typically lower resale vs SUVs/sedans.

2.2.2.2 Pair Plot for Correlation Analysis

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

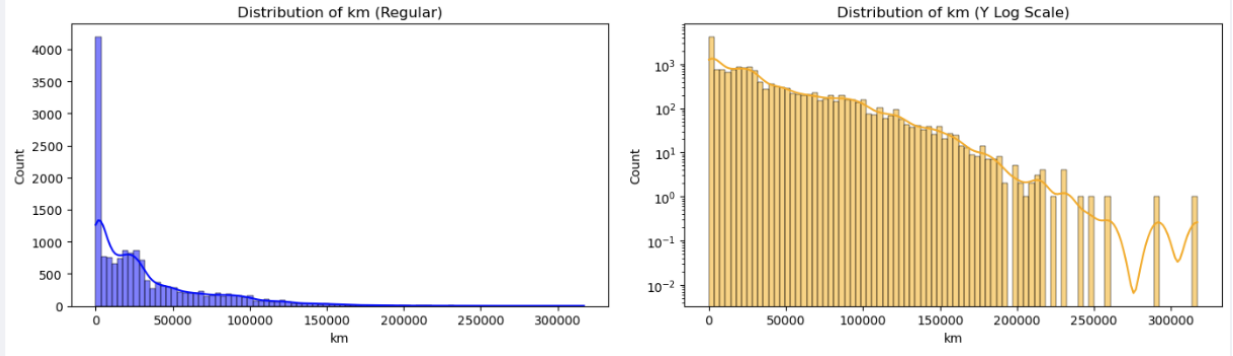


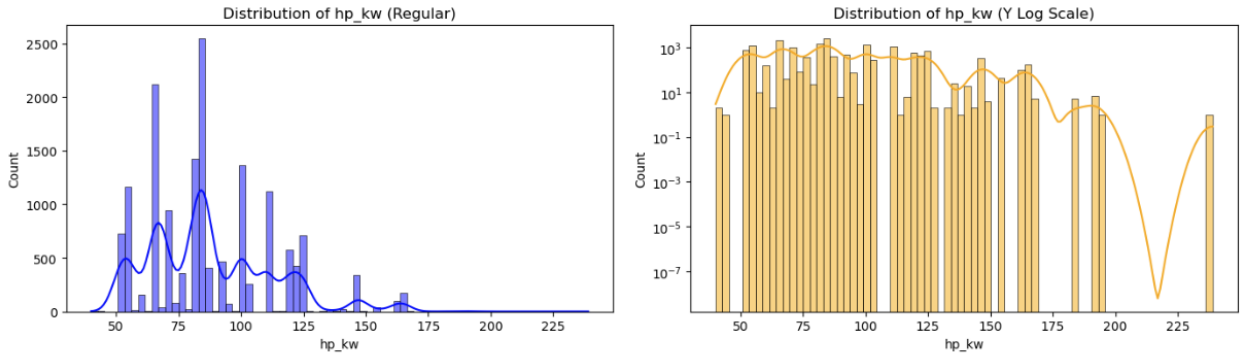
The pair plot confirms the correlations as identified earlier between numeric categorical variables and target ‘price_log’.

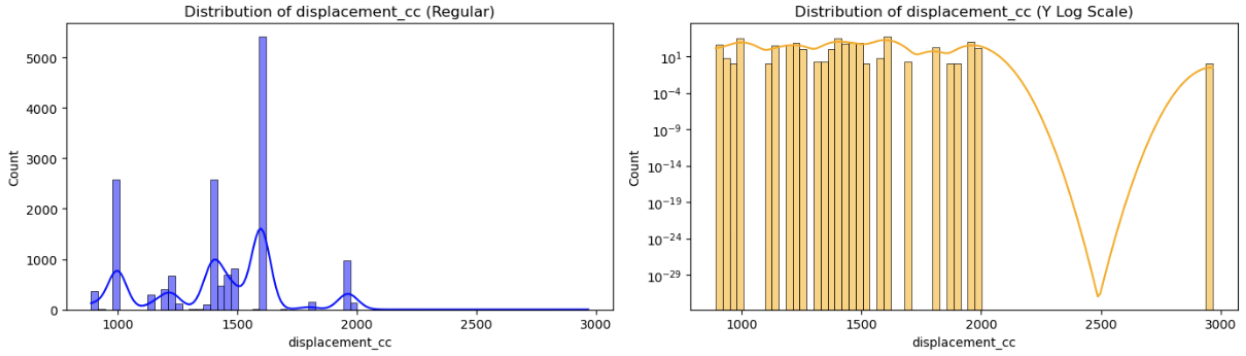
2.3 Outlier Analysis

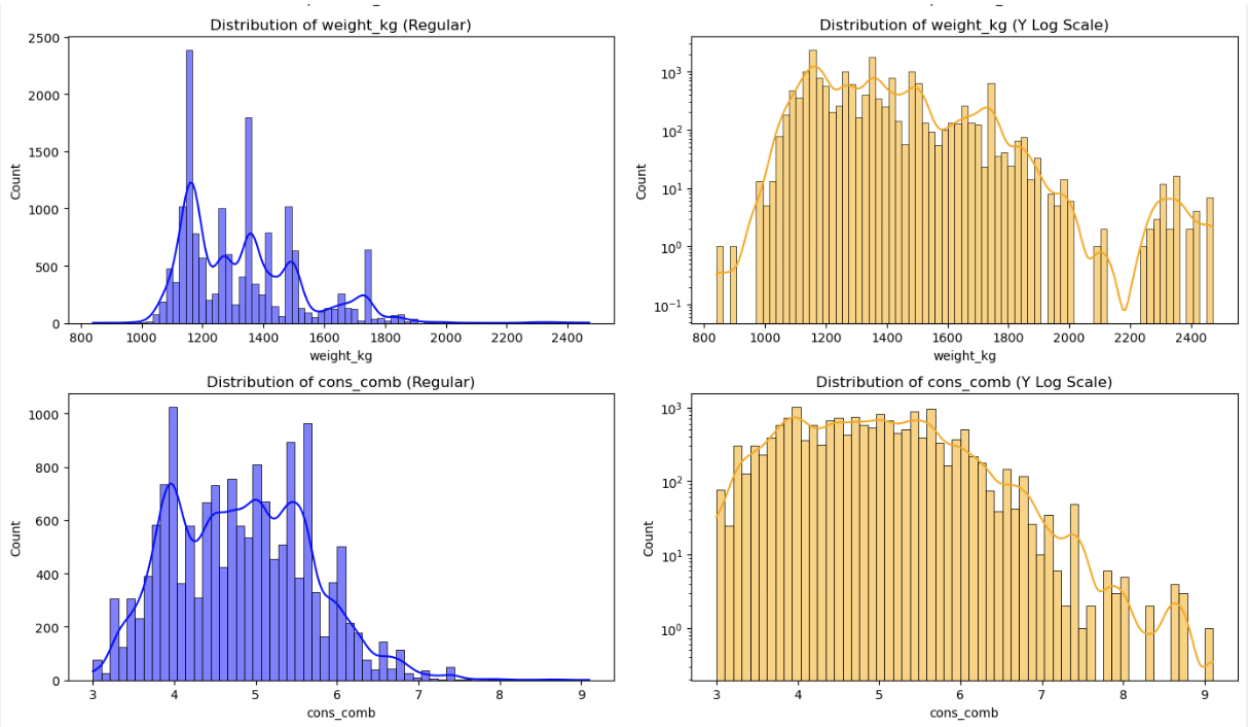
- Identify outliers in the numeric categorical variables
- Plot histogram with regular and log y-scale
- Plot box plot and violin plot

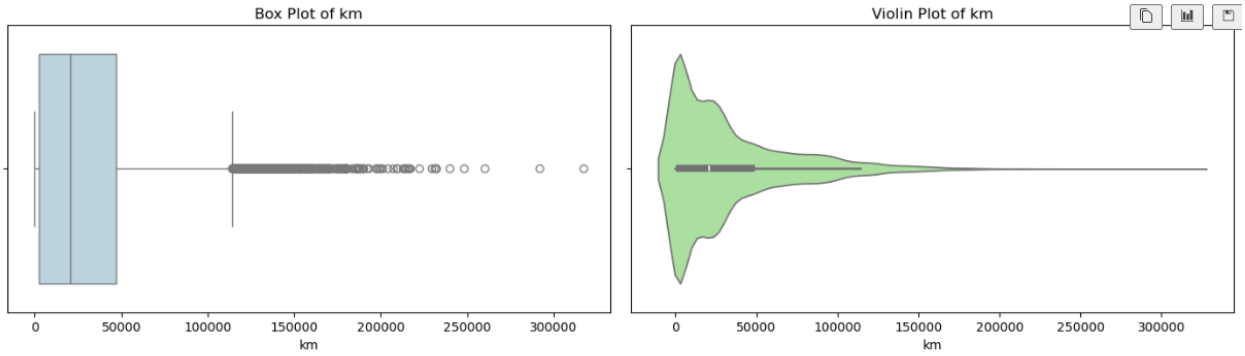
```
Pre Outlier Treatment on ['km', 'hp_kw', 'displacement_cc', 'weight_kg', 'cons_comb']:
```

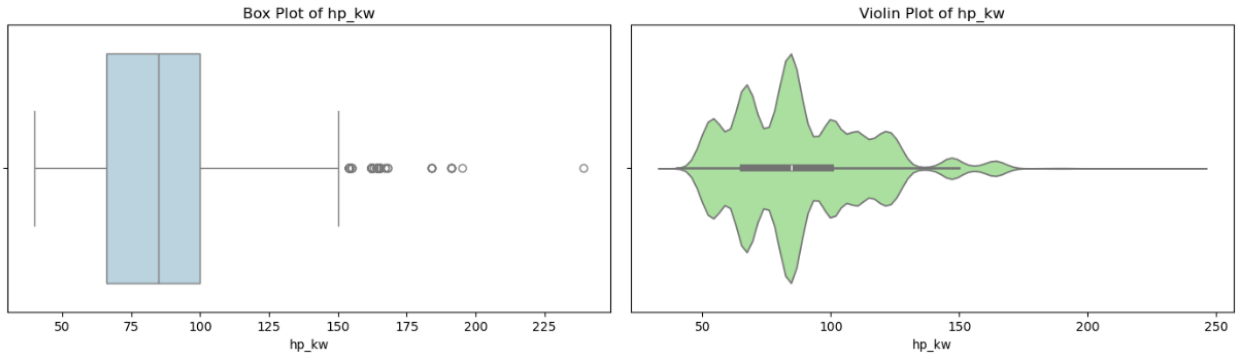


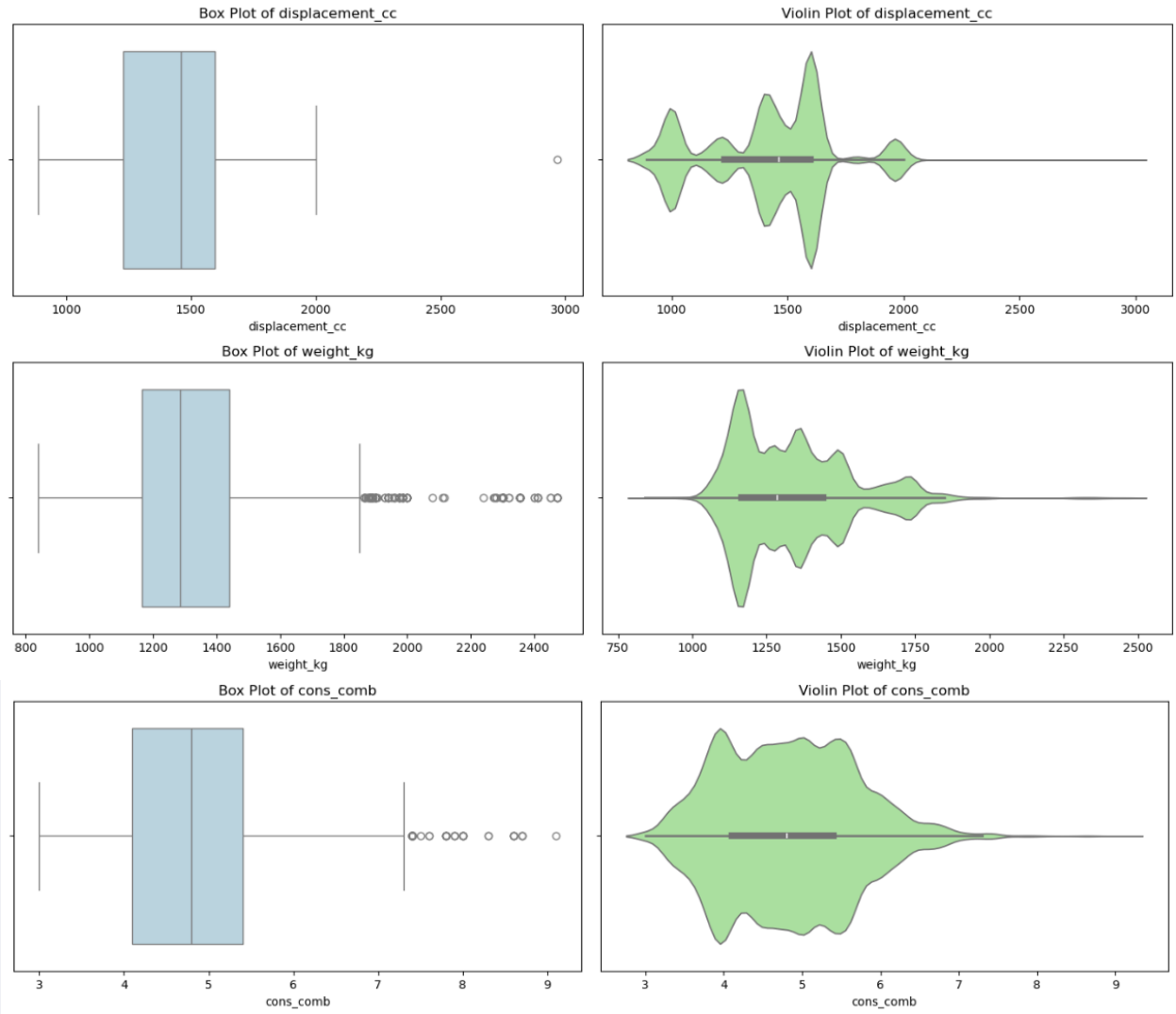





Distribution of weight_kg (Regular)
Count
weight_kg
Distribution of weight_kg (Y Log Scale)
Count
weight_kg
Distribution of cons_comb (Regular)
Count
cons_comb
Distribution of cons_comb (Y Log Scale)
Count
cons_comb


Box Plot of km
km
Violin Plot of km
km


Box Plot of hp_kw
hp_kw
Violin Plot of hp_kw
hp_kw

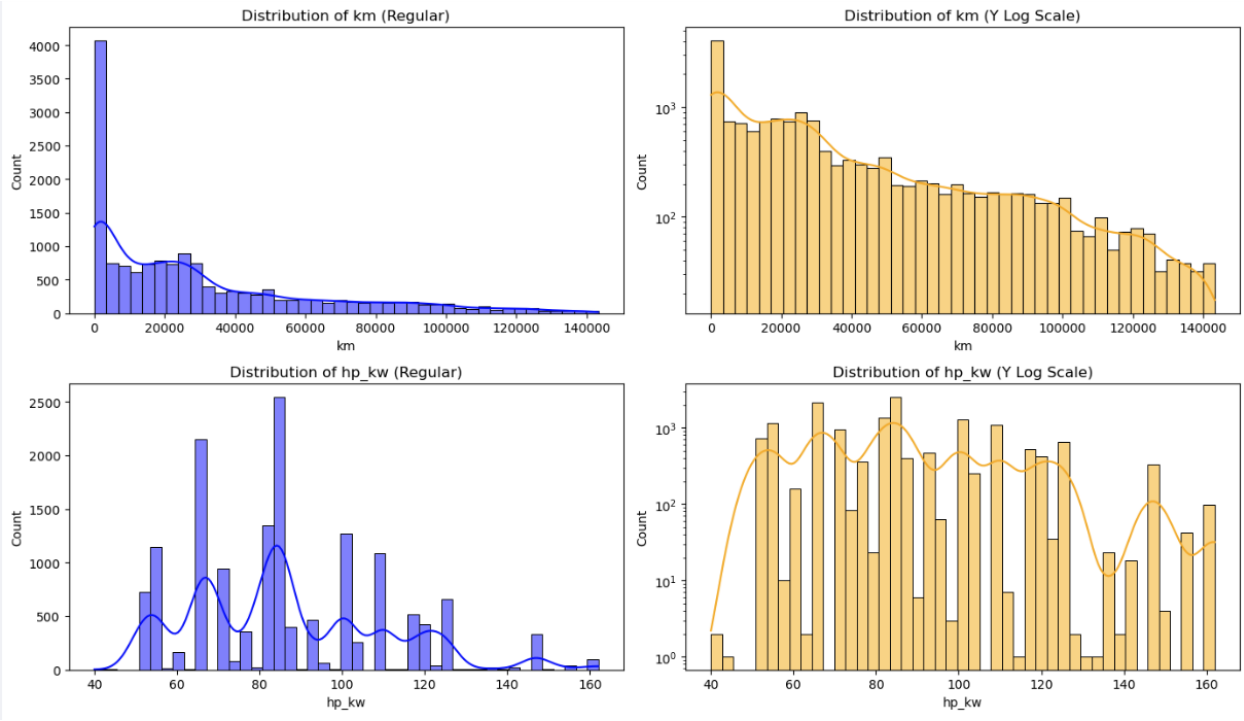


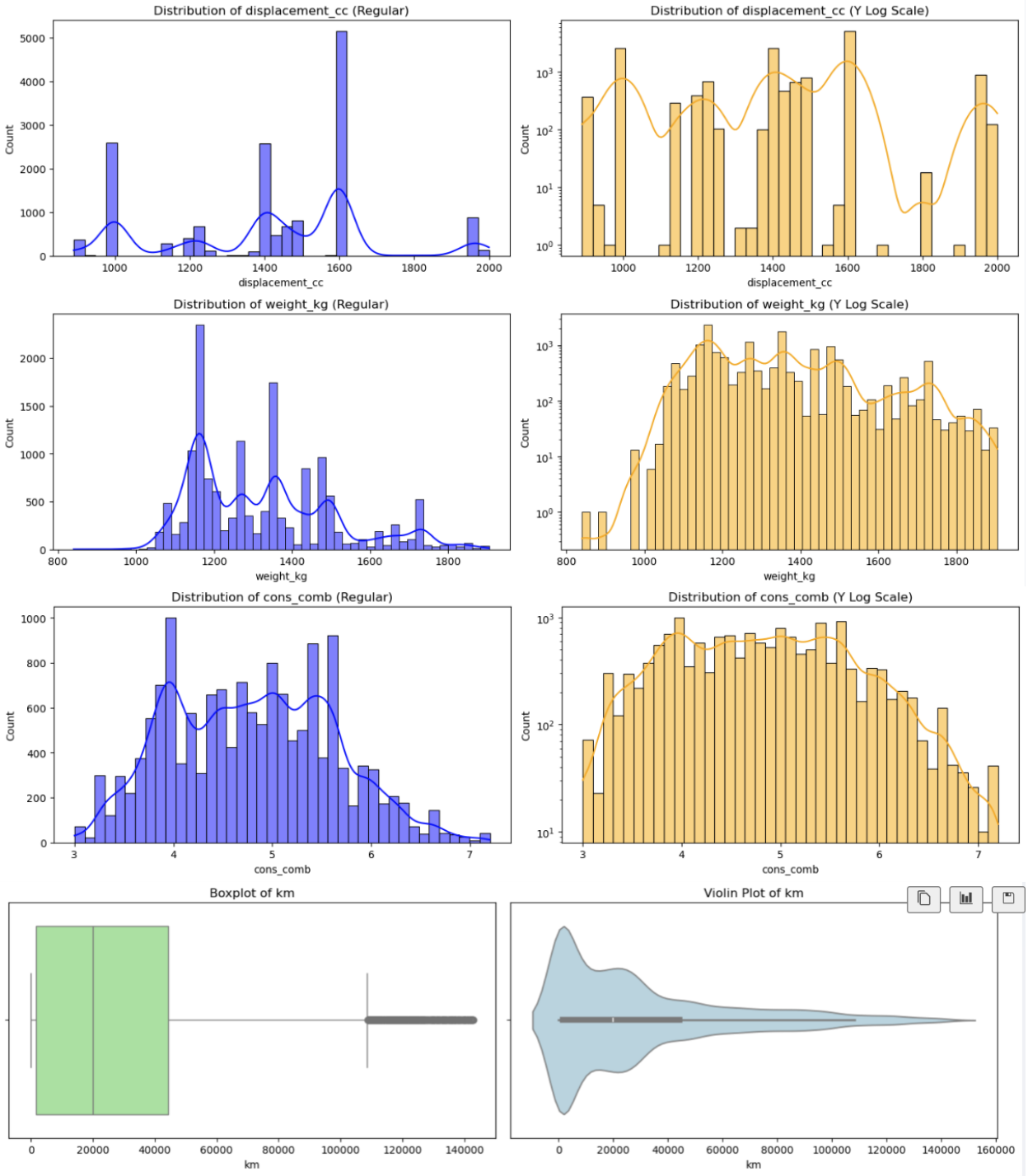
2.3.1 Handle Outliers in Numeric Categorical Variables

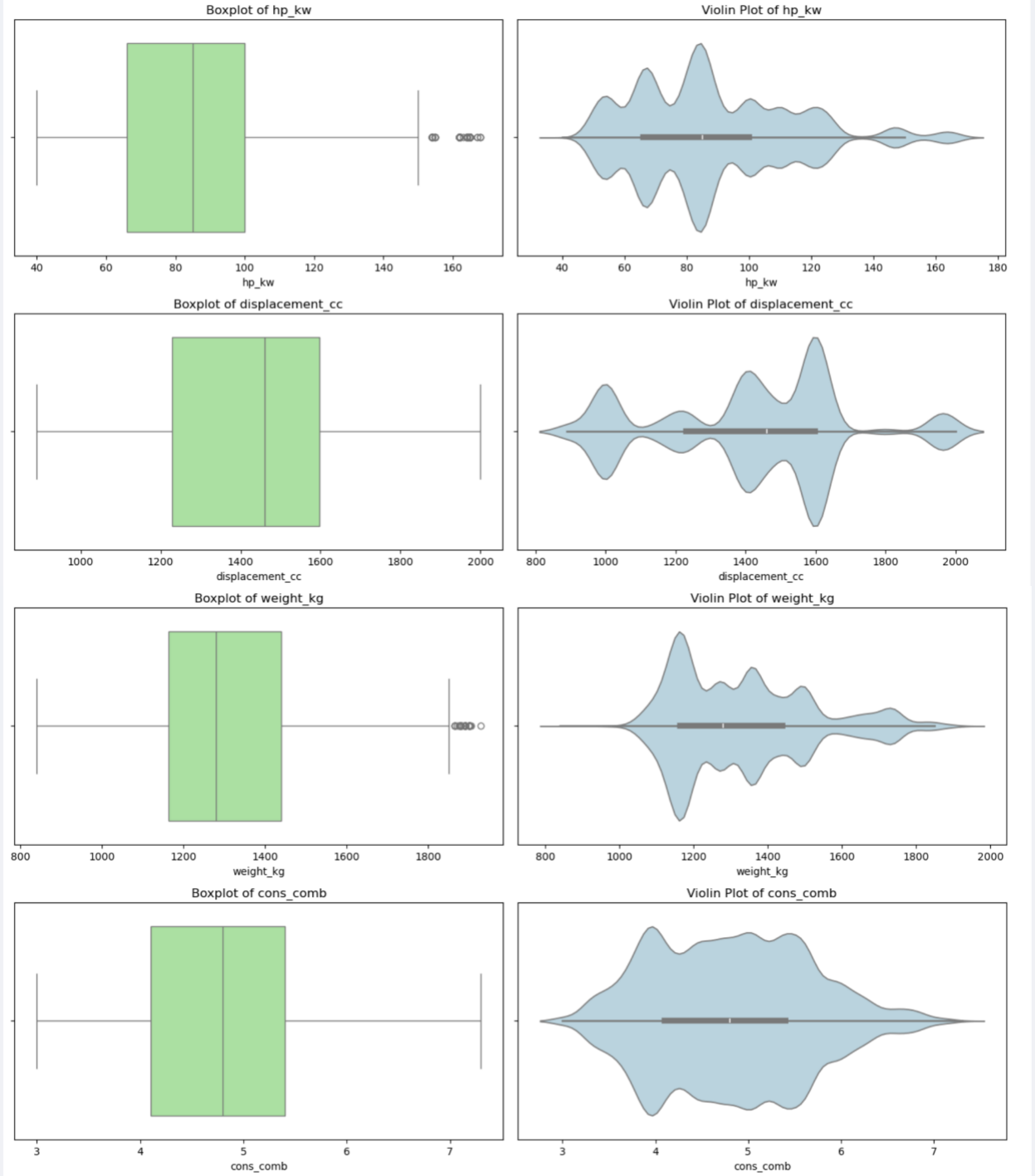
- Use z-score functioned defined earlier to identify outliers using threshold between 2 and 3
- Remove outliers, which are beyond 3 sigma deviation
- Plot histograms, box and violin charts to validate outcomes

```
--------------------------------------------------------------------------
Original record count: 15915
Number of outliers in 'km' using Z-score method with threshold 3: 262 (1.65%)
Data shape after removing outliers: (15653, 25)
Records removed: 262
--------------------------------------------------------------------------
Original record count: 15653
Number of outliers in 'hp_kw' using Z-score method with threshold 3: 35 (0.22%)
Data shape after removing outliers: (15618, 25)
Records removed: 35
--------------------------------------------------------------------------
Original record count: 15618
```

```
Number of outliers in 'displacement_cc' using Z-score method with threshold 3: 0 (0.00%)
Data shape after removing outliers: (15618, 25)
Records removed: 0
-------------------------------------------------------------------------------
Original record count: 15618
Number of outliers in 'weight_kg' using Z-score method with threshold 3: 86 (0.55%)
Data shape after removing outliers: (15532, 25)
Records removed: 86
-------------------------------------------------------------------------------
Original record count: 15532
Number of outliers in 'cons_comb' using Z-score method with threshold 3: 91 (0.59%)
Data shape after removing outliers: (15441, 25)
Records removed: 91
```




Distribution of displacement_cc (Regular)
Count
displacement_cc
Distribution of displacement_cc (Y Log Scale)
Count
displacement_cc
Distribution of weight_kg (Regular)
Count
weight_kg
Distribution of weight_kg (Y Log Scale)
Count
weight_kg
Distribution of cons_comb (Regular)
Count
cons_comb
Distribution of cons_comb (Y Log Scale)
Count
cons_comb
Boxplot of km
km
Violin Plot of km
km



2.3.1.1 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 4336 cars with zero age which is valid for new cars.
- There are 1851 cars with zero age and high km.
- There are 1557 cars with zero age and type <> new.
- There are 899 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 1827 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=1929.0, 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=167.0, missing=0, negative=0, zero=0
- cons_comb: min=3.0, max=7.3, missing=0, negative=0, zero=0
- age: min=0.0, max=3.0, missing=0, negative=0, zero=1827
- km: min=0.0, max=143000.0, missing=0, negative=0, zero=19
--------------------------------------------------------------------------
```

Check the outliers are realistic. Retain if valid and have real-world values. Check of records above 99th percentile for Weight, Displacement, HP and Fuel Consumption. Find out the mean values for these records, number of unique models, make model and check against real-world values.

```
Post Z-score Outlier Treatment - Records above 99th percentile:
--------------------------------------------------------------------------
- Weight (kg):
- Count of cars withWeight (kg) above 99th percentile: 110
- Average weight_kg of cars above 99th percentile: 1863.41 kg
- Number of Unique make_models above 99th percentile: 4
- Unique make_models above 99th percentile: ['Audi A3' 'Opel Astra' 'Opel Insignia'
'Renault Espace']
--------------------------------------------------------------------------
- Displacement (cc):
- Count of cars with Displacement (cc) above 99th percentile: 104
- Average displacement_cc of cars above 99th percentile: 1997.76 cc
- Number of Unique make_models above 99th percentile: 4
- Unique make_models above 99th percentile: ['Audi A1' 'Audi A3' 'Opel Insignia'
'Renault Espace']
--------------------------------------------------------------------------
- HP (kW):
- Count of cars with HP (kW) above 99th percentile: 113
- Average hp_kw of cars above 99th percentile: 164.76 kW
- Number of Unique make_models above 99th percentile: 2
- Unique make_models above 99th percentile: ['Renault Clio' 'Renault Espace']
--------------------------------------------------------------------------
- Fuel consumption (cons_comb):
- Count of cars with Fuel consumption (cons_comb) above 99th percentile: 51
- Average cons_comb of cars above 99th percentile: 7.05 L/100km
- Number of Unique make_models above 99th percentile: 3
- Unique make_models above 99th percentile: ['Opel Astra' 'Opel Corsa' 'Opel
Insignia']
--------------------------------------------------------------------------
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	>99th Percentile Summary	Interpretation	Verdict
Weight (kg)	~1,863 kg average among top 1%, seen in *Audi A3*, *Opel Astra*, *Insignia*, *Renault Espace*	Those are **mid-to-large sedans / MPVs**, which *naturally weigh 1.7–2.0 tons*. Nothing implausible here.	**Valid outliers** (keep)
Displacement (cc)	~1,998 cc avg, *Audi A1/A3*, *Insignia*, *Espace*	2.0 L engines are standard for premium trims. Definitely real data.	**Valid outliers** (keep)
HP (kW)	~165 kW (≈ 220 HP), *Renault Clio*, *Espace*	High-performance trims exist (e.g., *Clio RS*, *Espace GT*). Values are within market reality (< 300 HP).	**Valid outliers** (keep)
Fuel consumption (L/100 km)	~7.0 L/100 km, *Opel Astra/Corsa/Insignia*	7 L/100 km is *normal to slightly high*, not erroneous. May indicate sporty engines or urban cycles.	**Valid outliers** (keep)

2.4 Feature Engineering

2.4.1 Fix Redundant Columns and Create New Ones

- Create 3 new engineered features
    - Power_to_weight_ratio = hp_kw / weight_kg
    - Km_per_year = km / (age + 1)
    - Engine_efficiency = displacement_cc / hp_kw

```
Creating additional features:
- Created 'power_to_weight' feature as hp_kw / weight_kg
- power_to_weight stats: min=0.0321, max=0.1490, mean=0.0648
- Created 'km_per_year' feature as km / (age + 1) (if age > 0 else km)
- km_per_year stats: min=0.00, max=68000.00, mean=11181.03
- Created 'engine_efficiency' feature as displacement_cc / hp_kw
- engine_efficiency stats: min=7.1357, max=36.5250, mean=17.3879
We will retain the original columns as well for modeling and see the impact of
regularization techniques.
```

Feature	Formula	Intuition	Why it can help
power_to_weight	hp_kw / weight_kg	Captures engine strength per mass (performance indicator).	Combines two correlated predictors into one ratio — Lasso may pick it instead of both.
km_per_year	km / (age + 1)	Measures usage intensity.	Differentiates older low-mileage vs. newer high-mileage cars — useful for model signal clarity.
engine_efficiency	displacement_cc / hp_kw	Rough inverse of engine tuning.	Optional — may not improve model, but adds interpretability and diagnostic value.

While there are few highly correlated columns that can be removed, we will retain them for now and use Ridge and Lasso regression later

One-hot Encoding:

- 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: 12932 entries, 0 to 15913
Data columns (total 39 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   price                           12932 non-null  int64
 1   km                              12932 non-null  float64
 2   gears                           12932 non-null  float64
 3   comfort_convenience             12932 non-null  object
 4   entertainment_media             12932 non-null  object
 5   extras                          12932 non-null  object
 6   safety_security                 12932 non-null  object
 7   age                             12932 non-null  float64
 8   previous_owners                 12932 non-null  float64
```

```
 9   hp_kw                                12932 non-null  float64
 10  inspection_new                       12932 non-null  int64
 11  displacement_cc                      12932 non-null  float64
 12  weight_kg                            12932 non-null  float64
 13  cons_comb                            12932 non-null  float64
 14  price_log                            12932 non-null  float64
 15  power_to_weight                      12932 non-null  float64
 16  km_per_year                          12932 non-null  float64
 17  engine_efficiency                    12932 non-null  float64
 18  make_model_Audi A2                   12932 non-null  int64
 19  make_model_Audi A3                   12932 non-null  int64
...
 37  gearing_type_Semi-automatic          12932 non-null  int64
 38  drive_chain_front                    12932 non-null  int64
dtypes: float64(12), int64(23), object(4)
```

2.4.2 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: 27 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']}
```

2.4.3 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: (12932, 106)

<class 'pandas.core.frame.DataFrame'>
Index: 12932 entries, 0 to 15913
Columns: 106 entries, price to safety_security_Blind spot monitor
dtypes: float64(8), int64(93)
memory usage: 9.9 MB
```

2.4.4 Split Data into Training and Test

- Separate dependent and independent variables into X and y
- Retain target variable as price_log
- Retain original price column in a separate y_with_orig dataset for reference
- Create X_train, X_test, y_train_log, y_test_log dataframes using test size 0.2 and random state 42 (for consistency across runs)

```
Shape of X_train: (10345, 104), y_train: (10345,), X_test: (2587, 104), y_test:
(2587,)
```

Train-Test Split Inference:

- The dataset was split into training (80%) and testing (20%) subsets using the log-transformed target (price_log).
- Both price and price_log columns were removed from the feature matrix X to avoid leakage.
- The original price column was retained separately (y_with_orig) for evaluation and inverse transformation during metric computation.
- A fixed random seed ensures reproducibility.

2.4.5 Scale Features

2.4.5.1 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.

2.4.5.2 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.

2.4.5.3 Scale Features

- Select StandardScalar() approach
- Fit scalar transform using X_train dataset
- Transform X_train and X_test using the fitted scalar

```
Shape of X_train_scaled, X_test_scaled : (10345, 104), (2587, 104)
```

Scaling Inference:

- Features were standardized using Z-score scaling (StandardScaler) to ensure all predictors have comparable influence on the regression coefficients.
- The scaler was fit on the training data and applied to the test data to prevent data leakage.

3 Linear Regression Models

3.1 Baseline Linear Regression Model

3.1.1 Build Basic Linear Regression Model & Evaluate Performance

- Initialize LinearRegression() model
- Fit the model on X_train_scaled dataset
- Predict y_train_log_pred, y_test_log_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 –
    - Inverse transform the log values to get the y actual and predicted values
    - 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
--------------------------------------------------------------------------------
```

	feature	coef_log_scale	coef_abs	sign
15	make_model_Opel Corsa	-0.159893	0.159893	-
17	make_model_Renault Clio	-0.133566	0.133566	-
9	power_to_weight	0.128286	0.128286	+
14	make_model_Opel Astra	-0.081312	0.081312	-
2	age	-0.079387	0.079387	-
0	km	-0.064103	0.064103	-
30	gearing_type_Manual	-0.060522	0.060522	-
4	hp_kw	-0.059259	0.059259	-
7	weight_kg	0.056213	0.056213	+
18	make_model_Renault Espace	0.050818	0.050818	+

```
--------------------------------------------------------------------------------
Bottom 10 Coefficients with sign
--------------------------------------------------------------------------------
```

	feature	coef_log_scale	coef_abs	sign
76	extras_Roof rack	7.484882e-04	7.484882e-04	+
101	safety_security_Traffic sign recognition	5.053781e-04	5.053781e-04	+
88	safety_security_Tire pressure monitoring system	3.361097e-04	3.361097e-04	+
103	safety_security_Blind spot monitor	2.619943e-04	2.619943e-04	+
62	entertainment_media_On-board computer	-2.444862e-04	2.444862e-04	-
97	safety_security_LED Daytime Running Lights	-2.349946e-04	2.349946e-04	-
58	comfort_convenience_Electric tailgate	-2.084964e-04	2.084964e-04	-
37	comfort_convenience_Electrical side mirrors	4.403911e-05	4.403911e-05	+
93	safety_security_Alarm system	-3.265472e-05	3.265472e-05	-
12	make_model_Audi A2	-5.030698e-17	5.030698e-17	-

```
--------------------------------------------------------------------------------
Analyze Model Performance (on original scale) - Baseline Linear Regression
--------------------------------------------------------------------------------
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1890.889624	6323.232932	3.575464e+06	3.998327e+07	1210.026725	16884.875109	0.070142	0.910567
1	Test	1907.033451	6225.883135	3.636777e+06	3.876162e+07	1221.702503	16946.799768	0.070641	0.906140

3.1.1.1 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 R^2 on train and test means your model is both accurate and generalizes well.

Metric	Train	Test	Interpretation
RMSE	1,890	1,907	Average price prediction error ~1.9K — remarkably low relative to mean price (~16.9K). Indicates excellent fit.
MAE	1,210	1,222	Mean absolute error around 1.2K — predictions are tightly clustered around actual values.
R^2	0.91	0.906	Model explains ~91% of variance in price — very strong baseline performance.
MAPE	7.0%	7.1%	Predictions are within ±7% of actual price — highly acceptable for car pricing models.
Train vs Test	Close across all metrics		Indicates minimal overfitting — scaling, transformations, and encoding are working effectively.

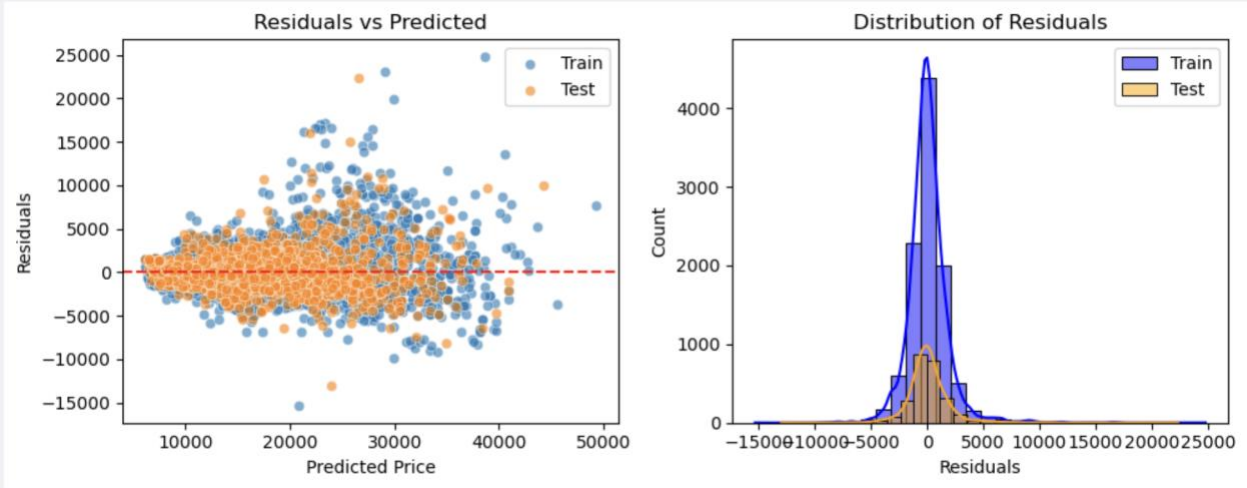
3.1.1.2 Summary:

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

3.1.2 Analyze Residuals

- Inverse transform to get original y values for y_train and y_test
- 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

```
Baseline Model - Plot of Residuals vs Predicted (on original scale):
--------------------------------------------------------------------------------
```



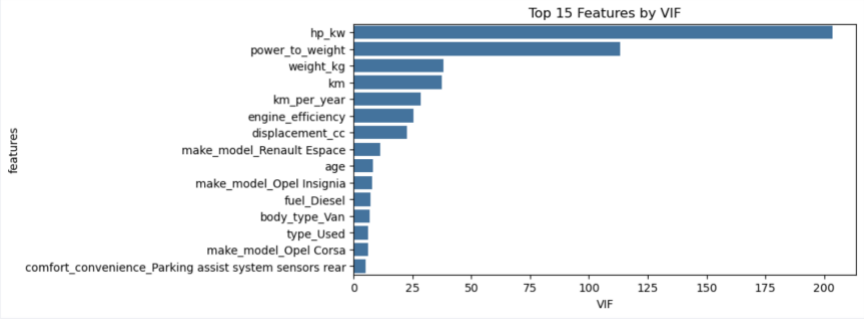
Residual Plots Interpretation:

- Mean of residuals is zero
- Spread looks symmetric (no skew)
- No clear pattern (e.g., residuals increasing only for certain predicted price ranges).
- Distribution of residuals is normal for train and test both

3.1.3 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 - R^2) from that regression.
    - High VIF (>5 or >10) means the feature is highly collinear with others



Variance Inflation Factor (VIF) for features:

	features	VIF
0	hp_kw	203.463609
1	power_to_weight	113.422714
2	weight_kg	38.132665
3	km	37.505943
4	km_per_year	28.616955
5	engine_efficiency	25.255668
6	displacement_cc	22.549607
7	make_model_Renault Espace	11.238599
8	age	8.312268
9	make_model_Opel Insignia	7.942626
10	fuel_Diesel	7.310767
11	body_type_Van	6.690555
12	type_Used	6.226696
13	make_model_Opel Corsa	6.147499
14	comfort_convenience_Parking assist system sens...	4.982086
15	cons_comb	4.689666
16	body_type_Station wagon	4.672582
17	make_model_Opel Astra	4.510686
18	make_model_Renault Clio	4.226622
19	comfort_convenience_Park Distance Control	4.224146

3.1.3.1 Interpretation

- VIF < 5 → Fine
- 5 ≤ VIF < 10 → Moderate multicollinearity, keep an eye
- VIF ≥ 10 → Strong multicollinearity, consider fixing

Feature	VIF	Interpretation

hp_kw	203.46	Extremely high — strongly collinear with power_to_weight, weight_kg, and displacement_cc. Represents core engine power relationship.
power_to_weight	113.42	Severe redundancy with both hp_kw and weight_kg. Derived feature amplifies existing relationship. Ridge/Lasso will handle.
weight_kg	38.13	High multicollinearity — tied to both horsepower and displacement; heavier cars generally have higher engine output.
km	37.51	Very high collinearity with age and km_per_year; these variables capture overlapping vehicle usage information.
km_per_year	28.62	Derived feature — strongly correlated with both age and km. Leave for now; regularization will address redundancy.
engine_efficiency	25.26	High — computed from hp_kw and displacement_cc; naturally interdependent. Retain for interpretability but watch VIF.
displacement_cc	22.55	High collinearity with hp_kw and engine_efficiency. Both capture aspects of engine capacity.
make_model_Renault Espace	11.24	High — overlaps with body_type_Van (most Espace are vans) and higher weights.
age	8.31	Moderate — correlates with km and km_per_year; expected due to vehicle usage linkage.
make_model_Opel Insignia	7.94	Moderate — overlaps with other Opel model dummies and related body types.
fuel_Diesel	7.31	Moderate — associated with larger engines (hp_kw, displacement_cc, cons_comb).
body_type_Van	6.69	Moderate — correlates with make_model_Renault Espace; redundancy expected.
type_Used	6.23	Moderate — correlates with age and km; used cars are typically older and have higher mileage.
make_model_Opel Corsa	6.15	Moderate — overlaps with other Opel dummies and smaller engine specs.
comfort_convenience_ Parking assist…	4.98	Acceptable — slight correlation with other comfort features (e.g., Park Distance Control).
cons_comb	4.69	Acceptable — moderate correlation with engine size and efficiency.
body_type_Station wagon	4.67	Acceptable — overlaps with specific models (Astra wagon, Renault Espace).
make_model_Opel Astra	4.51	Acceptable — moderate overlap with other Opel dummies and body types.
make_model_Renault Clio	4.23	Acceptable — some overlap with Renault brand effects but within safe limits.
comfort_convenience_ Park Distance Control	4.22	Acceptable — overlaps with other parking-assist-related features but low impact.

Muti-Collinearity Interpretation Summary:

- Multicollinearity is severe among engine-related and derived numerical features but moderate to low elsewhere.
- This is an expected outcome after introducing interaction-style engineered variables.
- No manual feature removal was performed — subsequent Ridge and Lasso regression models will regularize and shrink redundant predictors automatically, ensuring model stability and interpretability.

3.2 Ridge Regression Implementation

3.2.1 Define Alpha Values

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

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

3.2.2.1 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
    - Y_train_log_ridge_pred
    - Y_test_log_ridge_pred
- Use this model to identify the chosen alpha, which will later be used for narrowing the range

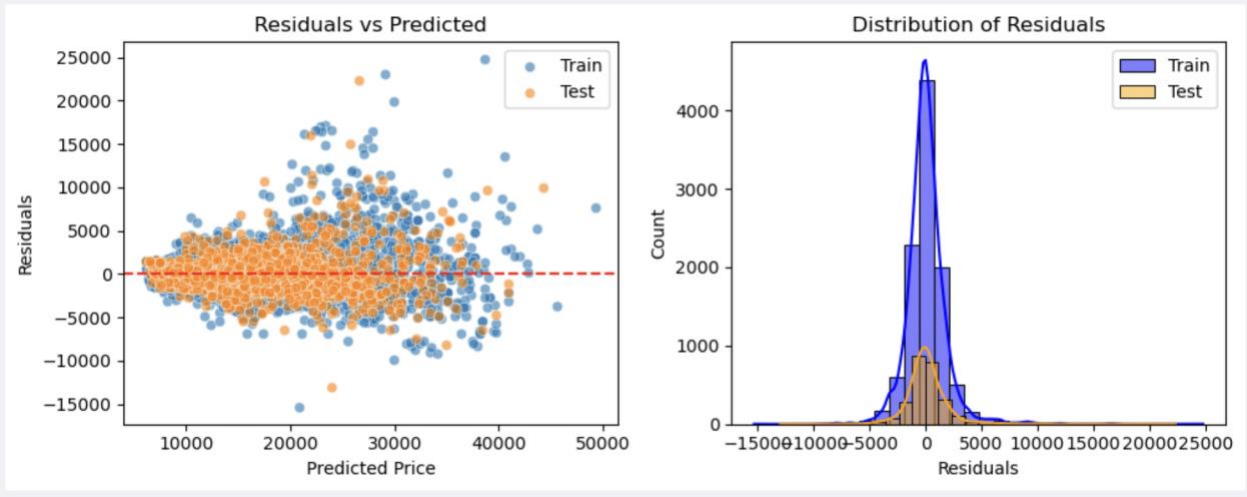
3.2.2.2 Analyze Model Performance & Identify Chosen Alpha

- 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

```
RidgeCV Model - Analyze Model Performance (on original scale):
------------------------------------------------------------------------
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1890.886842	6323.232932	3.575453e+06	3.998327e+07	1210.026185	16884.875109	0.070142	0.910568
1	Test	1907.034162	6225.883135	3.636779e+06	3.876162e+07	1221.702072	16946.799768	0.070640	0.906139

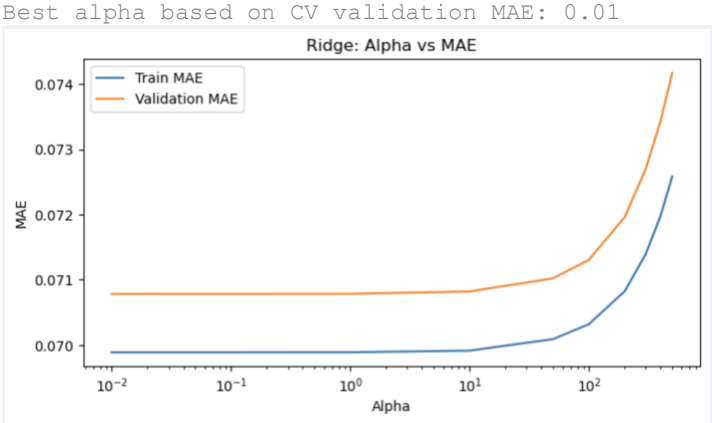
RidgeCV Model - Plot of Residuals vs Predicted (on original scale):



3.2.2.3 *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

Best alpha based on CV validation MAE: 0.01



3.2.2.4 *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): 0.01
```

We will get some best value of alpha above. This however is not the most accurate value but the best value from the given list. Now we have a rough estimate of the range that best alpha falls in. Let us do another iteration over the values in a smaller range.

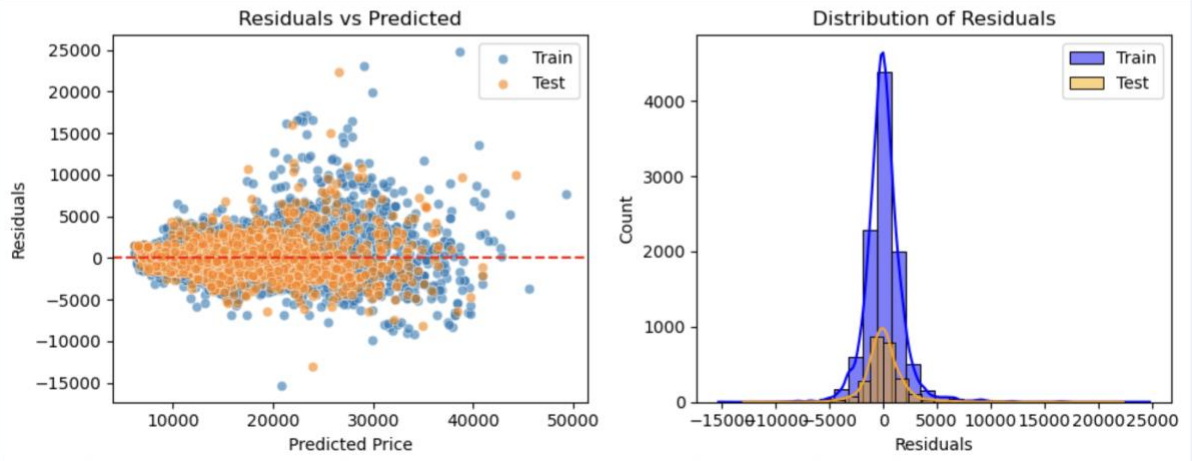
3.2.3 Fine Tune Alpha, Analyze Performance, Error-Alpha Chart, Coefficients

- We will narrow down the alpha range around the chosen alpha value from 3.2.2.4
- narrow_alpha_range = np.*linspace*(0.01, 10, 100) # 100 values between 0.01 and 10
- 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

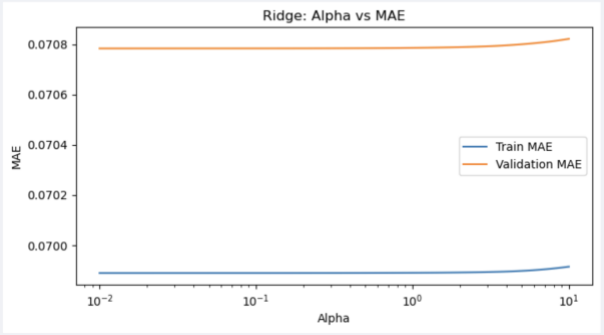
```
Final RidgeCV Model – Analyze Model Performance (on original scale):
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1890.886842	6323.232932	3.575453e+06	3.998327e+07	1210.026185	16884.875109	0.070142	0.910568
1	Test	1907.034162	6225.883135	3.636779e+06	3.876162e+07	1221.702072	16946.799768	0.070640	0.906139

```
Final RidgeCV Model – Plot of Residuals vs Predicted
Best RidgeCV Model Score (Negative MAE): 1221.7020722354073
Best RidgeCV Model Alpha: 0.01
```



```
Best alpha based on CV validation MAE: 0.01
```



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

```
--------------------------------------------------------------------------------
Final Ridge Model Coefficients with best score alpha=0.01
--------------------------------------------------------------------------------
Since the model was trained on log-transformed prices, the coefficients represent the
percentage change in price (approximately) for a one-standard-deviation increase in
the respective feature, holding others constant.
--------------------------------------------------------------------------------
Top 10 Coefficients with sign
--------------------------------------------------------------------------------
```

	feature	coef_log_scale	coef_abs	sign	approx_%_change
15	make_model_Opel Corsa	-0.159891	0.159891	-	-14.776324
17	make_model_Renault Clio	-0.133564	0.133564	-	-12.502876
9	power_to_weight	0.128259	0.128259	+	13.684731
14	make_model_Opel Astra	-0.081312	0.081312	-	-7.809406
2	age	-0.079387	0.079387	-	-7.631795
0	km	-0.064102	0.064102	-	-6.209081
30	gearing_type_Manual	-0.060521	0.060521	-	-5.872625
4	hp_kw	-0.059225	0.059225	-	-5.750547
7	weight_kg	0.056199	0.056199	+	5.780779
18	make_model_Renault Espace	0.050816	0.050816	+	5.212903

```
--------------------------------------------------------------------------------
Bottom 10 Coefficients with sign
--------------------------------------------------------------------------------
```

	feature	coef_log_scale	coef_abs	sign	approx_%_change
76	extras_Roof rack	0.000748	0.000748	+	0.074860
101	safety_security_Traffic sign recognition	0.000505	0.000505	+	0.050534
88	safety_security_Tire pressure monitoring system	0.000336	0.000336	+	0.033603
103	safety_security_Blind spot monitor	0.000262	0.000262	+	0.026162
62	entertainment_media_On-board computer	-0.000245	0.000245	-	-0.024453
97	safety_security_LED Daytime Running Lights	-0.000235	0.000235	-	-0.023516
58	comfort_convenience_Electric tailgate	-0.000209	0.000209	-	-0.020878
37	comfort_convenience_Electrical side mirrors	0.000044	0.000044	+	0.004379
93	safety_security_Alarm system	-0.000033	0.000033	-	-0.003257
12	make_model_Audi A2	0.000000	0.000000	-	0.000000

3.2.4 Evaluate the Ridge Model on Test Data

- Display the summary values from test data model performance

```
Final Test Evaluation:
R²: 0.9061
MAE: 1221.70
MSE: 3636779.29
RMSE: 1907.03
```

3.2.5 Ridge Model Interpretation Summary

Metric	Train	Test	Interpretation
RMSE	1890.9	1907.0	Same as baseline — stable fit
MAE	1210.0	1221.7	Identical to OLS
R^2	0.911	0.906	Identical to OLS

Validation behavior: Validation MAE increased beyond $\alpha \approx 10$, confirming that higher regularization leads to underfitting.

Inference:

- Ridge regularization did not improve predictive performance, which indicates:
  - The data has no severe overfitting issue.
  - Multicollinearity is present (seen in high VIF values), but it doesn't harm predictive accuracy significantly.
- Ridge provides coefficient stability rather than performance gains.

3.3 Lasso Regression Implementation

3.3.1 Define Alpha Range

We will start with alpha range of [0.01, 0.1, 1, 10, 20]

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

3.3.2.1 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.

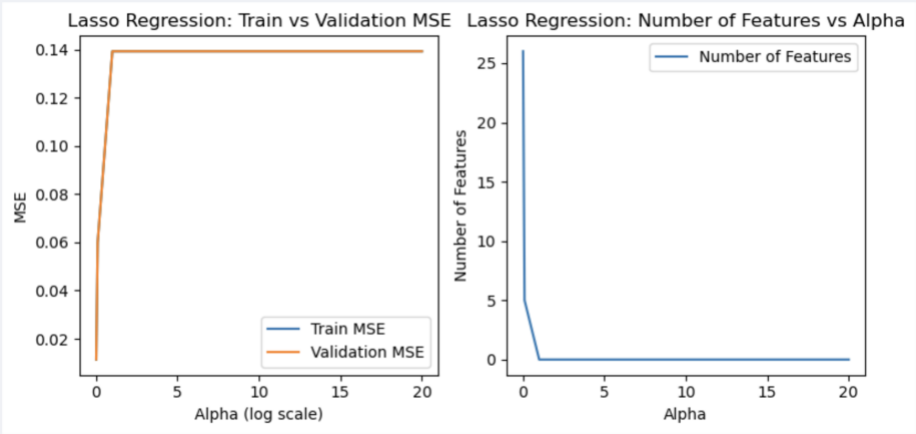
- For each value of alpha in the range
  - Initialize Lasso() model
  - Fit the model on training data
  - Predict the y target on training and test data both (use the log values)
- Find the train and test scores using mean_squarred_error
- Apply Lasso Regularisation and find the best value of alpha from the list

Score Type	Computed on	Purpose
train_scores	Training data	Measures how well the model fits the data it was trained on.
val_scores	Cross-validation folds (inside training set)	Evaluates how well the model generalizes during hyperparameter tuning.
test_scores	Independent test set	Assesses final model performance on truly unseen data.

3.3.2.2 *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.

```
Train MSE: [0.01139, 0.05998, 0.13925, 0.13925, 0.13925]
Validation MSE:   [0.01146, 0.06002, 0.13926, 0.13926, 0.13926]
```



Interpretation:

- Train and Validation MSE almost overlap, thus looking like a single line
- Number of features drops to zero quickly with alpha close to 1

3.3.2.3 *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): 0.01
Chosen alpha (Lasso) with minimum Test MSE: 0.01, Test MSE: 0.011708
```

3.3.3 Fine Tune with Narrow Alpha Range

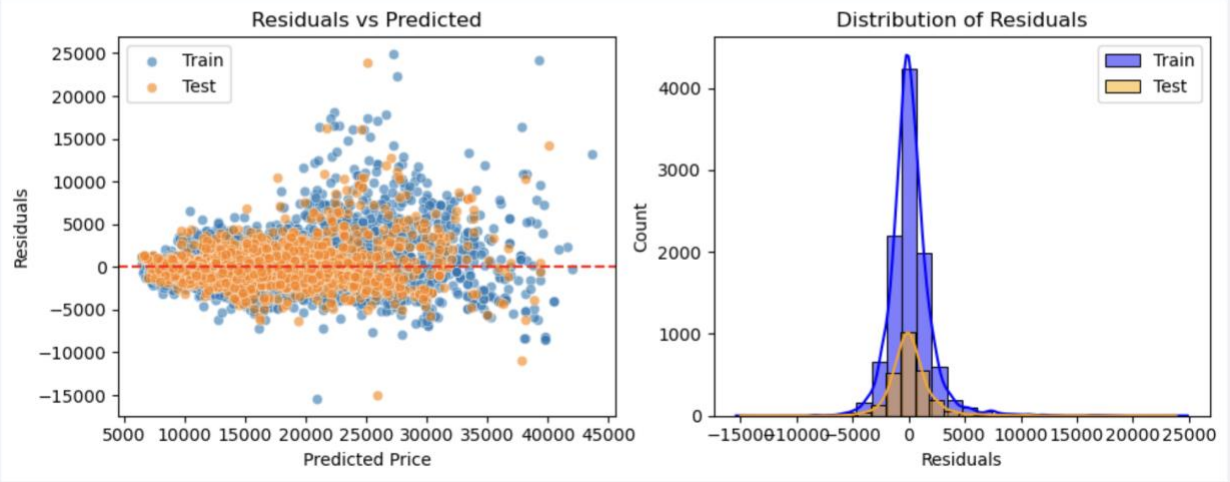
- We use narrow alpha range : np.linspace(0.005, 0.1, 30) # 30 values between 0.005 and 10
- Fit a LassoCV model to the training data and returns predictions for both train and test sets. Also returns the chosen alpha value
- Initialize LassoCV with narrow alpha range and *cv=KFold(n_splits=5, shuffle=True, 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: y_train_lasso_log_pred, y_test_lasso_log_pred
- Inverse transform the predictions to get y values on original scale
- 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

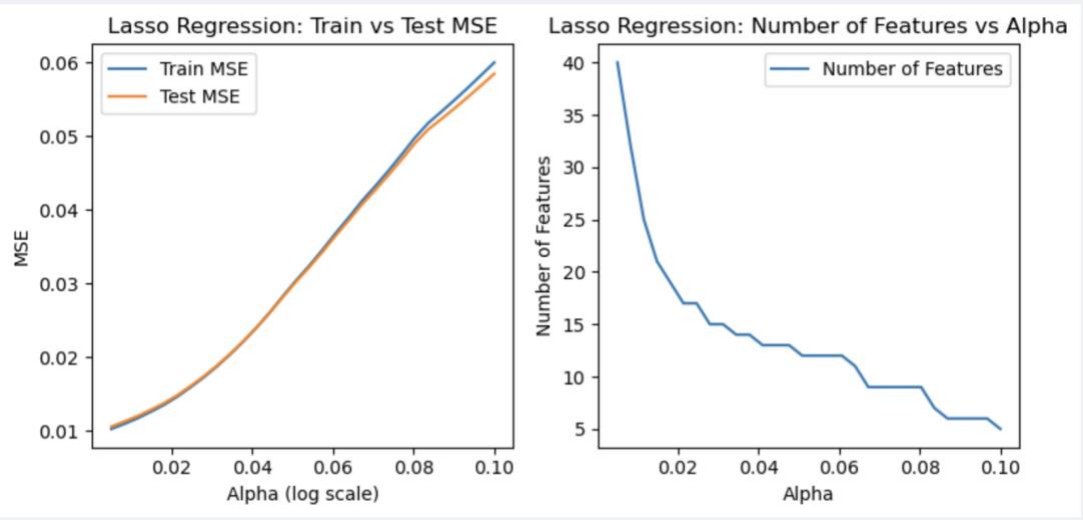
```
Final LassoCV Model - Analyze Model Performance (on original scale):
Best alpha (using LassoCV): 0.005
Test MAE (original scale): 1301.57, RMSE: 2056.93, MSE: 4230953.61
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	2014.354420	6323.232932	4.057624e+06	3.998327e+07	1288.382295	16884.875109	0.074671	0.898507
1	Test	2056.928196	6225.883135	4.230954e+06	3.876162e+07	1301.573021	16946.799768	0.075093	0.890805

```
Final LassoCV Model – Plot of Residuals vs Predicted (on original scale):
```



Plot the train and test scores using lasso() tuning and min negative MSE



3.3.3.1 *Run the Model using Best Alpha*

```
Best alpha (using LassoCV): 0.005
Test MAE (original scale): 1301.57, RMSE: 2056.93, MSE: 4230953.61
Final LassoCV Model - Analyze Model Performance (on original scale):
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	2014.354420	6323.232932	4.057624e+06	3.998327e+07	1288.382295	16884.875109	0.074671	0.898507
1	Test	2056.928196	6225.883135	4.230954e+06	3.876162e+07	1301.573021	16946.799768	0.075093	0.890805

```
Since the model was trained on log-transformed prices, the coefficients represent the
percentage change in price (approximately) for a one-standard-deviation increase in
the respective feature, holding others constant.
Since the model was trained on log-transformed prices, the coefficients represent the
percentage change in price (approximately) for a one-standard-deviation increase in
the respective feature, holding others constant.
--------------------------------------------------------------------------------
Top 10 Lasso features and coefficients, with sign
```

	feature	coef_log_scale	coef_abs	sign	approx_%_change	dropped
15	make_model_Opel Corsa	-0.137035	0.137035	-	-12.806070	False
17	make_model_Renault Clio	-0.108059	0.108059	-	-10.242544	False
2	age	-0.093138	0.093138	-	-8.893251	False
30	gearing_type_Manual	-0.063029	0.063029	-	-6.108418	False
14	make_model_Opel Astra	-0.058836	0.058836	-	-5.713882	False
18	make_model_Renault Espace	0.051623	0.051623	+	5.297885	False
4	hp_kw	0.051275	0.051275	+	5.261201	False
13	make_model_Audi A3	0.047919	0.047919	+	4.908592	False
10	km_per_year	-0.034344	0.034344	-	-3.376048	False
0	km	-0.033002	0.033002	-	-3.246368	False

```
--------------------------------------------------------------------------------
Bottom 10 Lasso features and coefficients, with sign
--------------------------------------------------------------------------------
```

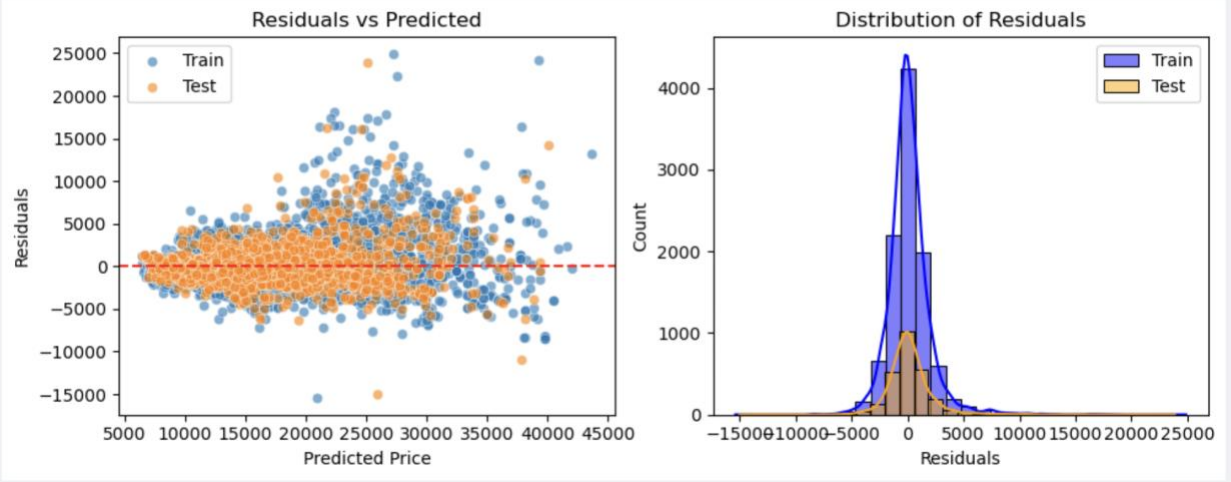
	feature	coef_log_scale	coef_abs	sign	approx_%_change	dropped
57	comfort_convenience_Electrically adjustable seats	0.0	0.0	0	0.0	True
53	comfort_convenience_Keyless central door lock	0.0	0.0	0	0.0	True
51	comfort_convenience_Parking assist system sens...	0.0	0.0	0	0.0	True
50	comfort_convenience_Tinted windows	0.0	0.0	0	0.0	True
49	comfort_convenience_Lumbar support	0.0	0.0	0	0.0	True
48	comfort_convenience_Start-stop system	0.0	0.0	0	0.0	True
45	comfort_convenience_Power windows	-0.0	0.0	0	0.0	True
44	comfort_convenience_Parking assist system sens...	0.0	0.0	0	0.0	True
41	comfort_convenience_Multi-function steering wheel	0.0	0.0	0	0.0	True
103	safety_security_Blind spot monitor	0.0	0.0	0	0.0	True

Dropped Lasso features and coefficients

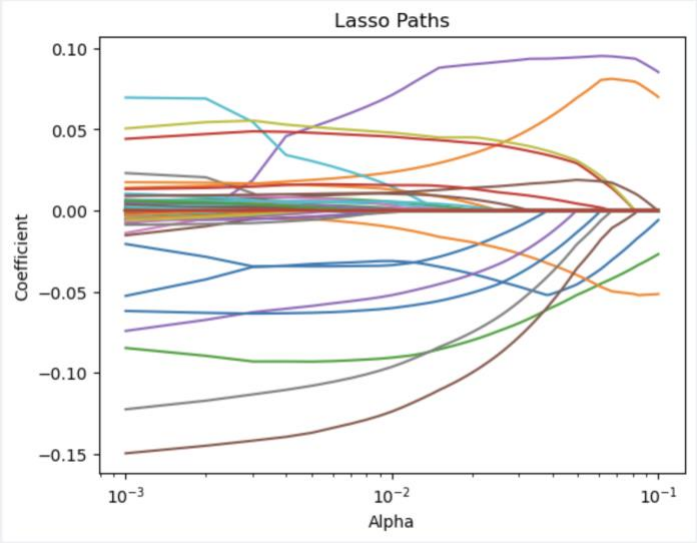
Total features dropped (coeff=0): 64 out of 104

	feature	coef_log_scale	coef_abs	sign	approx_%_change	dropped
87	safety_security_Side airbag	0.0	0.0	0	0.0	True
99	safety_security_Emergency brake assistant	0.0	0.0	0	0.0	True
70	extras_Catalytic Converter	0.0	0.0	0	0.0	True
71	extras_Voice Control	-0.0	0.0	0	0.0	True
72	extras_Sport seats	0.0	0.0	0	0.0	True
...	...	...	...	...	...	...
48	comfort_convenience_Start-stop system	0.0	0.0	0	0.0	True
45	comfort_convenience_Power windows	-0.0	0.0	0	0.0	True
44	comfort_convenience_Parking assist system sens...	0.0	0.0	0	0.0	True
41	comfort_convenience_Multi-function steering wheel	0.0	0.0	0	0.0	True
103	safety_security_Blind spot monitor	0.0	0.0	0	0.0	True

Final LassoCV Model – Plot of Residuals vs Predicted



Lasso Coefficient Path Visualization
visualize how coefficients shrink as α increases:



3.3.4 Evaluate Lasso Model on Test Data

- Print the test data evaluation summary

```
Final Test Evaluation:
R²: 0.8908
MAE: 1301.57
MSE: 4230953.61
RMSE: 2056.93
```

3.3.5 Lasso Model Inference Summary:

Metric	Training	Test	Interpretation
RMSE	2014.35	2056.93	Prediction error ~2000 on average, consistent across train/test → minimal overfitting.
MAE	1288.38	1301.57	Typical prediction error ≈ 1300, showing good generalization.
R^2	0.8985	0.8908	Model explains ~89% of price variability — excellent for a sparse linear model.

Interpretation Summary

- Lasso regression delivered a highly interpretable, parsimonious model that retained ~40 key predictors explaining 89% of the price variation.
- It successfully eliminated redundant comfort and safety features while maintaining strong predictive performance comparable to Ridge and Linear models.
- Dominant predictors include model type (Opel, Renault, Audi), age, horsepower, gear type, and annual mileage, all intuitively aligned with real-world car valuation drivers.

Best α (regularization strength): 0.005

a) → A relatively low alpha, meaning only mild regularization was required.
b) The model still benefits from L1 shrinkage while retaining key predictors.
c) MSE (Test): 4.23M, consistent with Ridge and Linear results — confirming stable variance–bias balance.

Feature Selection Behavior:

- Total features in model: 104
- Features retained (non-zero coefficients): 40
- Features dropped (set to zero by L1 penalty): 64

This confirms that Lasso performed strong feature selection, removing ~61% of predictors. Most dropped variables were comfort, extras, and safety features, which are either:

- Highly correlated with brand/model, or
- Low-variance binary indicators that don't independently explain price variation.

3.4 Regularization Comparison & Analysis

3.4.1 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

	Model	Alpha	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Linear	-	Training	1890.889624	6323.232932	3.575464e+06	3.998327e+07	1210.026725	16884.875109	0.070142	0.910567
1	Linear	-	Test	1907.033451	6225.883135	3.636777e+06	3.876162e+07	1221.702503	16946.799768	0.070641	0.906140
2	RidgeCV	0.01	Training	1890.886842	6323.232932	3.575453e+06	3.998327e+07	1210.026185	16884.875109	0.070142	0.910568
3	RidgeCV	0.01	Test	1907.034162	6225.883135	3.636779e+06	3.876162e+07	1221.702072	16946.799768	0.070640	0.906139
4	LassoCV	0.005	Training	2014.354420	6323.232932	4.057624e+06	3.998327e+07	1288.382295	16884.875109	0.074671	0.898507
5	LassoCV	0.005	Test	2056.928196	6225.883135	4.230954e+06	3.876162e+07	1301.573021	16946.799768	0.075093	0.890805

3.4.1.1 Interpretation of Combined Metrics

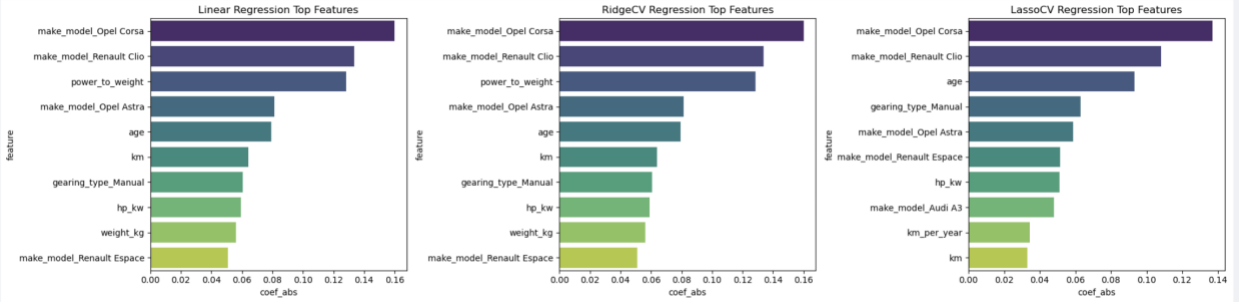
- All three models perform exceptionally well, achieving $R^2 \approx 0.89–0.91$, meaning they explain ~90% of the variance in car prices.
- RMSE values around 1,900–2,000 indicate that the models predict car prices within roughly ±2,000 on average.
- MAE is between 1,200–1,300, meaning the typical deviation between predicted and actual prices is around 6–8% of the mean car price (~17,000).
- The near-identical training and test metrics across all models show excellent generalization — no signs of overfitting or underfitting.
- 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.

3.4.2 Compare Coefficients of the 3 models

- Use the model_coeff datasets created in the functions for the respective 3 models
- Merge to show in a tabular format
    - Add % Impact for Linear, Ridge, Lasso Coeffs : Coefficients converted into approximate percentage change in price, using: (exp(coef) - 1) * 100
    - Add % Average Coeff across the 3 models and % Average Impact
- 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

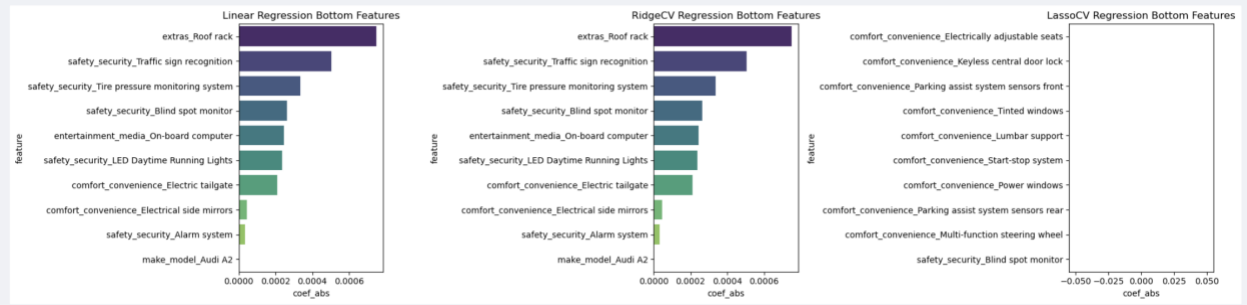
```
Top 10 Coefficients Comparison (Linear, Ridge, Lasso)
% Average Impact = Coefficients converted into approximate percentage change in price,
using: (exp(coef) - 1) * 100
```

	feature	Linear %Impact	Ridge %Impact	Lasso %Impact	Avg %Impact	Abs Avg %Impact
7	make_model_Opel Corsa	-14.78	-14.78	-12.81	-14.12	14.12
8	make_model_Renault Clio	-12.50	-12.50	-10.24	-11.75	11.75
10	power_to_weight	13.69	13.68	0.00	9.12	9.12
0	age	-7.63	-7.63	-8.89	-8.05	8.05
6	make_model_Opel Astra	-7.81	-7.81	-5.71	-7.11	7.11
1	gearing_type_Manual	-5.87	-5.87	-6.11	-5.95	5.95
9	make_model_Renault Espace	5.21	5.21	5.30	5.24	5.24
3	km	-6.21	-6.21	-3.25	-5.22	5.22
11	weight_kg	5.78	5.78	0.00	3.85	3.85
2	hp_kw	-5.75	-5.75	5.26	-2.08	2.08
5	make_model_Audi A3	0.00	0.00	4.91	1.64	1.64
4	km_per_year	0.00	0.00	-3.38	-1.13	1.13

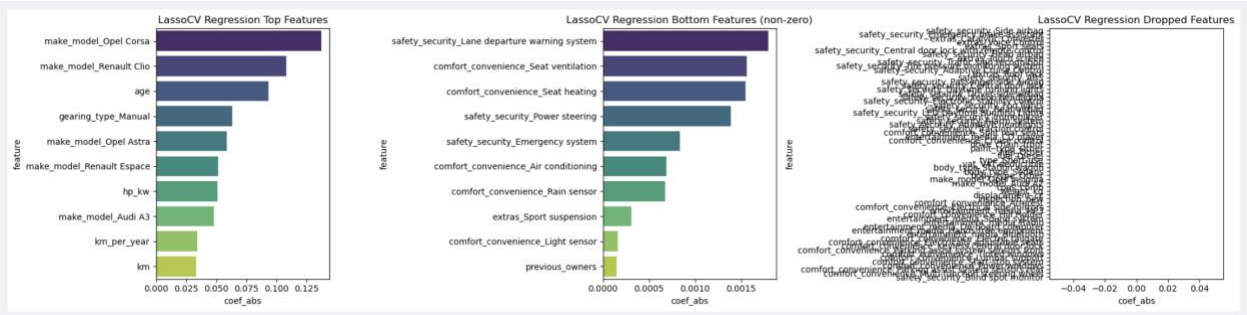



```
Bottom 10 Coefficients Comparison (Linear, Ridge, Lasso)
```

	feature	Linear Coef	Ridge Coef	Lasso Coef
0	extras_Roof rack	0.0007	0.0007	0.0
1	safety_security_Traffic sign recognition	0.0005	0.0005	0.0
2	safety_security_Blind spot monitor	0.0003	0.0003	0.0
3	safety_security_Tire pressure monitoring system	0.0003	0.0003	-0.0
4	safety_security_LED Daytime Running Lights	-0.0002	-0.0002	0.0
5	comfort_convenience_Electric tailgate	-0.0002	-0.0002	0.0
6	entertainment_media_On-board computer	-0.0002	-0.0002	0.0
7	extras_Touch screen	0.0000	0.0000	0.0
8	extras_Voice Control	0.0000	0.0000	-0.0
9	make_model_Audi A2	-0.0000	0.0000	0.0
10	comfort_convenience_Electrical side mirrors	0.0000	0.0000	0.0
11	extras_Sport seats	0.0000	0.0000	0.0
12	safety_security_Central door lock with remote ...	0.0000	0.0000	-0.0
13	safety_security_Emergency brake assistant	0.0000	0.0000	0.0
14	safety_security_Head airbag	0.0000	0.0000	-0.0
15	extras_Catalytic Converter	0.0000	0.0000	0.0
16	safety_security_Side airbag	0.0000	0.0000	0.0
17	safety_security_Alarm system	-0.0000	-0.0000	0.0



Lasso Model – Top 10, Bottom 10, and Dropped Coefficients (Side-by-Side)



4 Conclusion

4.1 Model Performance Comparison Conclusion

Model	Alpha	RMSE	MSE	MAE	MAPE	R^2
Linear	–	1907.03	3.6368e+06	1221.70	0.0706	0.9061
RidgeCV	0.01	1907.03	3.6368e+06	1221.70	0.0706	0.9061
LassoCV	0.005	2056.93	4.2309e+06	1301.57	0.0751	0.8908

4.1.1 Observations

- **Overall**:
  - All three models perform very closely, with Linear and Ridge being nearly identical in predictive accuracy:
  - RMSE ≈ 1907, MAE ≈ 1222, R^2 ≈ 0.906.
  - LassoCV performs slightly lower in accuracy but gains interpretability by pruning irrelevant features.
  - → Predictive accuracy is effectively the same, with R^2 ≈ between 0.89 and 0.91 for all models.
- **Baseline Linear Regression**
  - Already strong: R^2 ≈ 0.906, MAE ≈ 1222.
  - Regularization adds little to predictive accuracy but helps with stability and interpretability when features are correlated.
- **RidgeCV**
  - Chosen **α = 0.01**
  - Metrics nearly identical to baseline linear regression.
  - Regularization doesn't change predictions much but:
  - Stabilizes coefficients.
  - Slightly reduces overfitting and multicollinearity risk.
  - Best suited when you want to **retain all features** but make them **less sensitive to correlation**.
- **LassoCV**
  - Chosen **α = 0.005**
  - Predictive performance slightly lower (R^2 drops from **0.906 → 0.891**) but:
  - Performs **automatic feature selection** — dropped **64 of 104 features**.
  - Model becomes leaner, easier to interpret, and computationally efficient.
  - Excellent choice when **simplification and interpretability** are priorities.

4.1.2 Model Behaviour Summary

Aspect	Linear	Ridge	Lasso
Regularization Strength	None	Weak (α = 0.01)	Weak (α = 0.005)
Feature Shrinkage	No	Mild	Strong
Feature Elimination	No	No	Yes

Multicollinearity Control	No	Yes	Yes
Interpretability	Moderate	Moderate	High
Predictive Accuracy (R^2)	0.906	0.906	0.891

4.1.3 Model Performance Comparison 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).

4.1.4 Predictors and Coefficients Comparison

4.1.4.1 Top Predictors across Models

Feature	Linear Coef	Ridge Coef	Lasso Coef	Interpretation
make_model_Opel Corsa	-0.1599	-0.1599	-0.1370	Budget model → strong negative impact.
make_model_Renault Clio	-0.1336	-0.1336	-0.1081	Negative impact — entry-level, lower resale.
power_to_weight	0.1283	0.1283	0.0000	Performance proxy — strong in Linear/Ridge, dropped in Lasso (redundant with hp).
age	-0.0794	-0.0794	-0.0931	Older cars lose value; consistent across models.
make_model_Opel Astra	-0.0813	-0.0813	-0.0588	Moderate negative impact.
gearing_type_Manual	-0.0605	-0.0605	-0.0630	Manual cars valued lower.
km	-0.0641	-0.0641	-0.0330	More mileage → lower resale.
make_model_Renault Espace	0.0508	0.0508	0.0516	Family model → mild positive value.
weight_kg	0.0562	0.0562	0.0000	Dropped by Lasso (collinear with hp_kw).
hp_kw	-0.0593	-0.0592	0.0513	Power drives price, but interaction effects differ.

4.1.5 Bottom Predictors across Models

Feature	Linear Coef	Ridge Coef	Lasso Coef	Interpretation
extras_Roof rack	0.0007	0.0007	0.0	Insignificant accessory, no resale effect.

safety_security_Traffic sign recognition	0.0005	0.0005	0.0	Negligible premium.
safety_security_Blind spot monitor	0.0003	0.0003	0.0	No measurable impact.
safety_security_LED Daytime Running Lights	-0.0002	-0.0002	0.0	Cosmetic only, ignored by Lasso.
comfort_convenience_Electric tailgate	-0.0002	-0.0002	0.0	Minimal influence on price.
extras_Touch screen	0.0000	0.0000	0.0	Common feature, no price differentiation.
extras_Voice Control	0.0000	0.0000	-0.0	Negligible or redundant.
make_model_Audi A2	-0.0000	0.0000	0.0	Irrelevant; dropped by Lasso.
safety_security_Central door lock...	0.0000	0.0000	-0.0	Basic equipment, zero effect.
extras_Catalytic Converter	0.0000	0.0000	0.0	Standardized; zero importance.

4.1.6 Lasso Simplification

61% of the low impact features dropped by Lasso, simplifying the model interpretation. Examples of **Lasso Dropped Example Features:**

- extras_Voice Control
- comfort_convenience_Power windows
- safety_security_Blind spot monitor
- extras_Sport seats
- comfort_convenience_Start-stop system

These are redundant or low-impact features — Lasso confirms they don't contribute meaningfully once stronger predictors (hp, age, mileage, brand/model) are included.

Category	Count
Total features	104
Retained (non-zero)	40
Dropped (coeff = 0)	64

4.1.6.1 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).

4.1.6.2 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).

4.1.6.3 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.

4.1.6.4 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.**

4.1.6.5 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., `comfort convenience related`) 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 ($R^2 \approx 0.895$-0.906).
  - **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**