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 regularisation techniques.

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

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

Data Dictionary

Variable	Description
`make_model`	The brand and model of the vehicle (e.g., 'Audi A1').
`body_type`	The body style of the vehicle, such as Sedan, Compact, or Station Wagon.
`price`	The listed price of the car in currency.
`vat`	Indicates the VAT status for the vehicle's price (e.g., VAT deductible, Price negotiable).
`km`	The total mileage (in kilometers) of the vehicle, indicating its usage.
`Type`	Condition of the vehicle, whether it's 'Used' or 'New'.
`Fuel`	Type of fuel the vehicle uses, such as 'Diesel', 'Benzine', etc.
`Gears`	The number of gears in the vehicle's transmission.
`Comfort_Convenience`	Comfort and convenience features, such as 'Air conditioning', 'Leather steering wheel', 'Cruise control', and more.
`Entertainment_Media`	Media features available in the vehicle, including 'Bluetooth', 'MP3', 'Radio', etc.
`Extras`	Additional features like 'Alloy wheels', 'Sport suspension', etc.
`Safety_Security`	Safety features like 'ABS', 'Airbags', 'Electronic stability control', 'Isofix', etc.
`age`	Age of the car (calculated based on the model year).
`Previous_Owners`	The number of previous owners the car has had.
`hp_kW`	Engine power in kilowatts (kW), indicating the performance capacity of the engine.
`Inspection_new`	Indicates whether the car has recently undergone an inspection (1 for yes, 0 for no).
`Paint_Type`	The type of paint on the car, such as 'Metallic', 'Matte', etc.
`Upholstery_type`	The material used for the interior upholstery, such as 'Cloth', 'Leather', etc.
`Gearing_Type`	The type of transmission the car uses, either 'Automatic' or 'Manual'.
`Displacement_cc`	The engine displacement in cubic centimeters (cc), indicating the size of the engine.
`Weight_kg`	The total weight of the vehicle in kilograms.
`Drive_chain`	The type of drivetrain, indicating whether it's 'Front' or 'Rear' wheel drive.
`cons_comb`	The combined fuel consumption in liters per 100 kilometers.

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

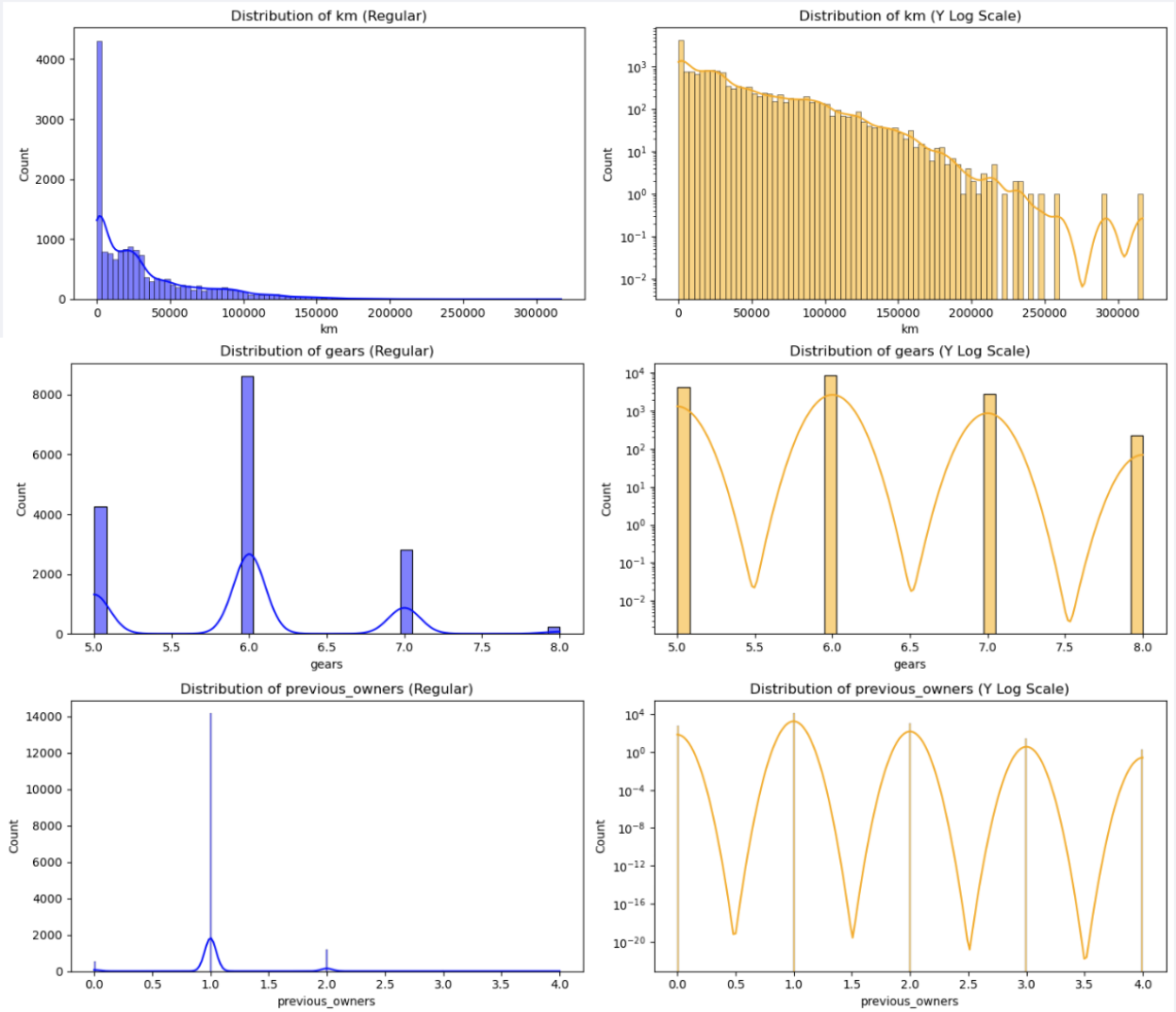
Results:

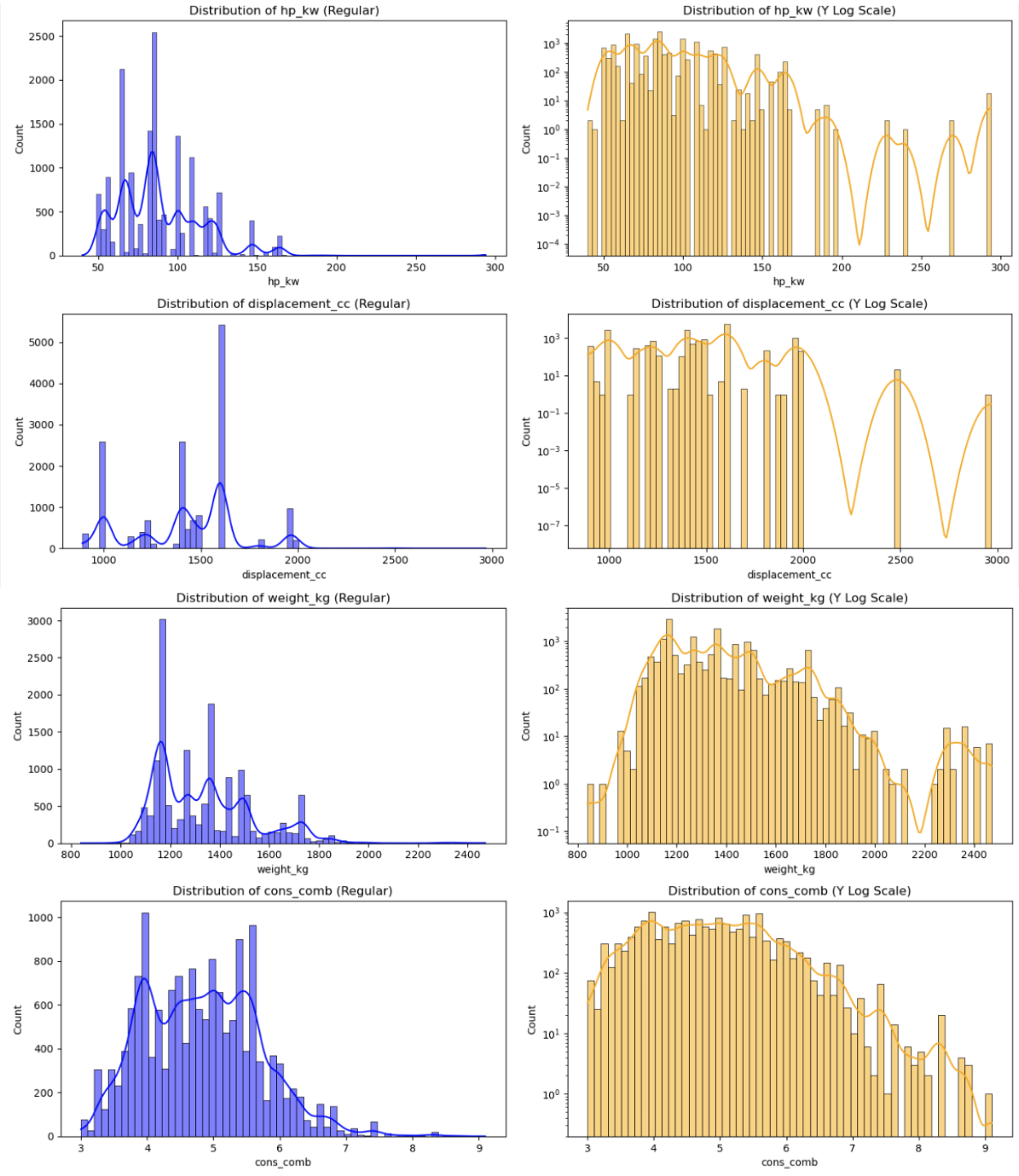
Column Name	Data Type	Missing Count	Missing %	Impute Type	Action
make_model	object	0	0.0	mode	none
age	float64	0	0.0	median	none
drive_chain	object	0	0.0	mode	none
weight_kg	float64	0	0.0	mean	none
displacement_cc	float64	0	0.0	mean	none
gearing_type	object	0	0.0	mode	none
upholstery_type	object	0	0.0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

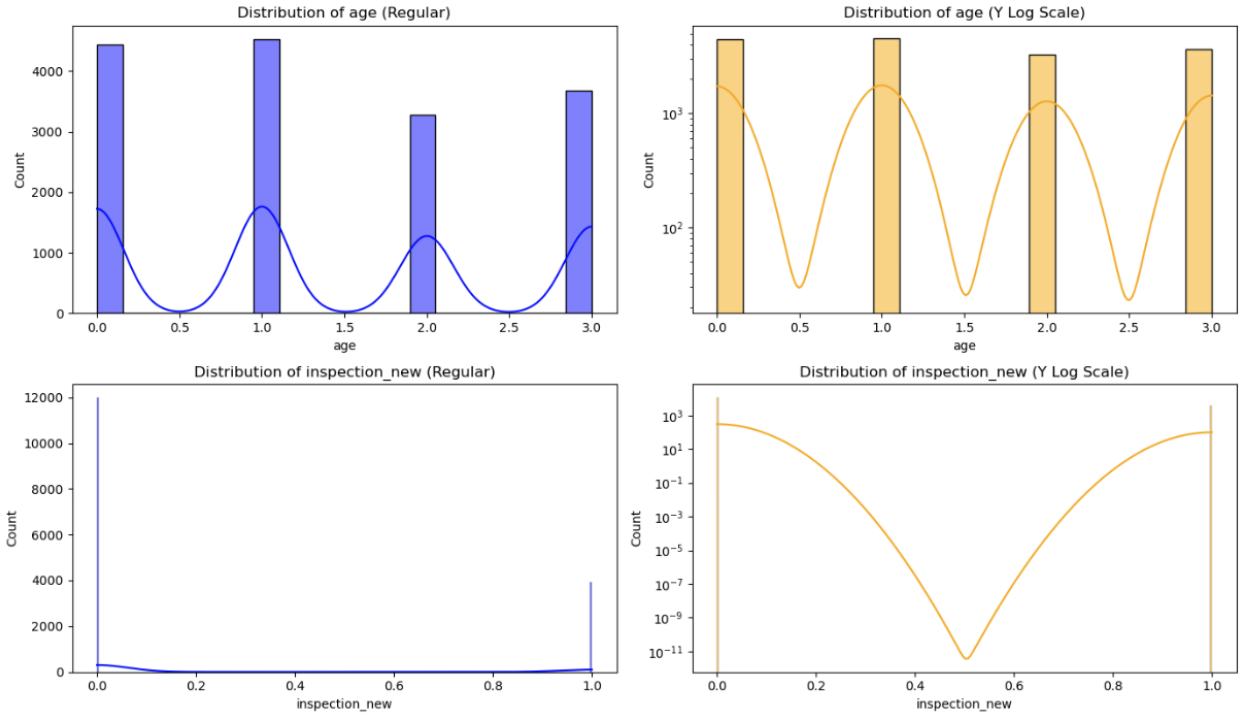
entertainment_media	object	0	0.0	mode	none
comfort_convenience	object	0	0.0	mode	none
gears	float64	0	0.0	mode	none
fuel	object	0	0.0	mode	none
type	object	0	0.0	mode	none
km	float64	0	0.0	median	none
vat	object	0	0.0	mode	none
price	int64	0	0.0	drop	none
cons_comb	float64	0	0.0	mean	none

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'

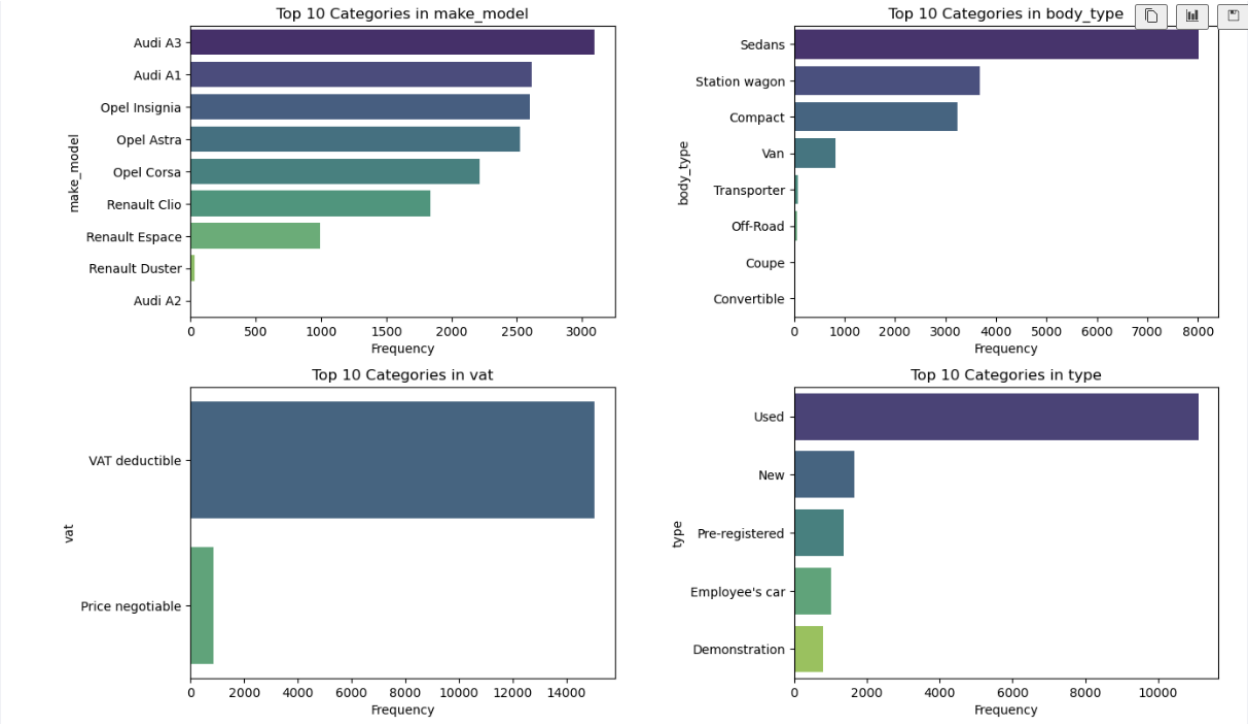


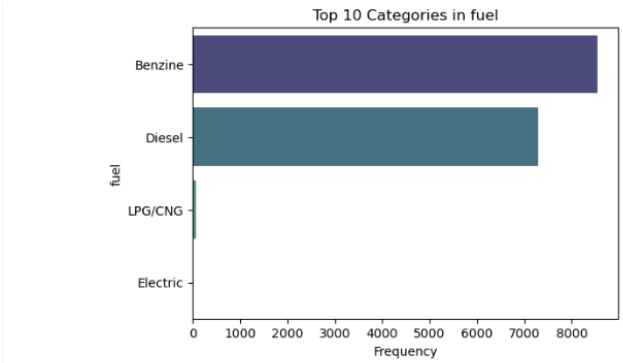

Distribution of hp_kw (Regular)
Count
hp_kw
Distribution of hp_kw (Y Log Scale)
Count
hp_kw
Distribution of displacement_cc (Regular)
Count
displacement_cc
Distribution of displacement_cc (Y Log Scale)
Count
displacement_cc
Distribution of weight_kg (Regular)
Count
weight_kg
Distribution of weight_kg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

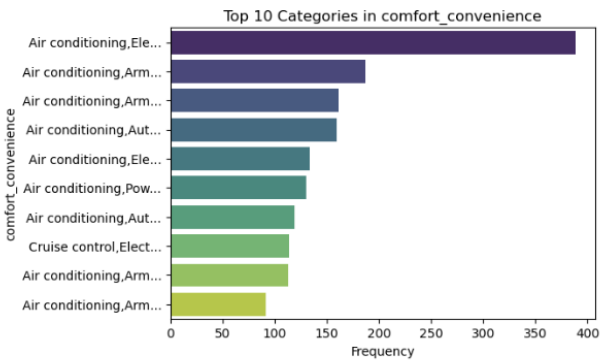


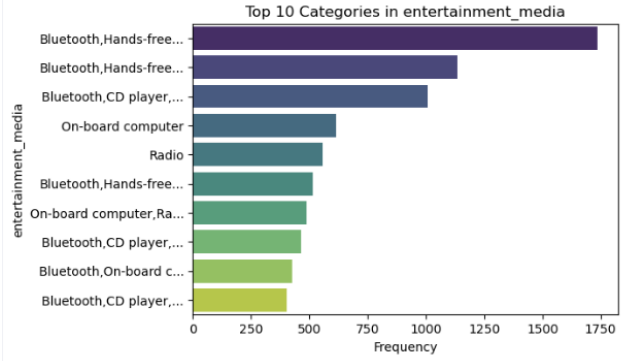
2.1.3 Idendity Categorical Predictors and Plot Frequency Distributions

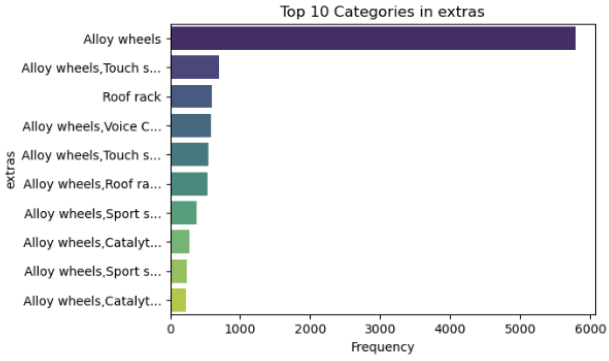
- Plot bar chart for each categorical column, with counts on x-axis & top-10 categories on the y-axis

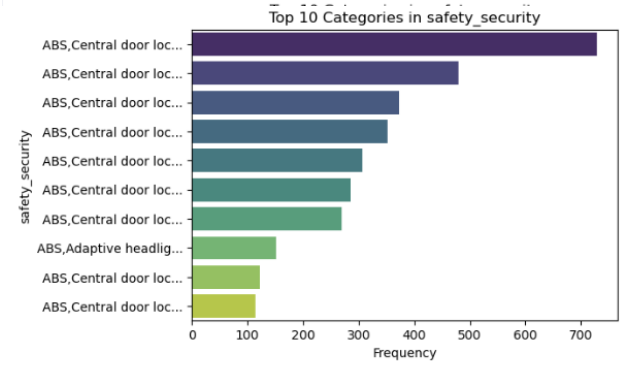


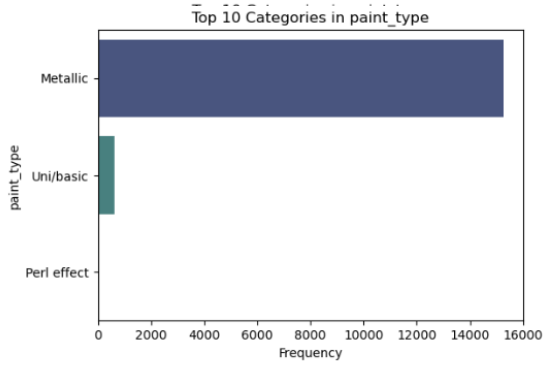

Top 10 Categories in fuel
Benzine
Diesel
LPG/CNG
Electric
fuel
0
1000
2000
3000
4000
5000
6000
7000
8000
Frequency

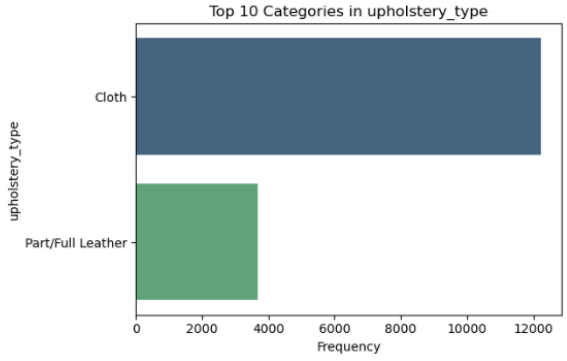

Top 10 Categories in comfort_convenience
Air conditioning,Ele...
Air conditioning,Arm...
Air conditioning,Arm...
Air conditioning,Aut...
Air conditioning,Ele...
Air conditioning,Pow...
Air conditioning,Aut...
Cruise control,Elect...
Air conditioning,Arm...
Air conditioning,Arm...
comfort_convenience
0
50
100
150
200
250
300
350
400
Frequency

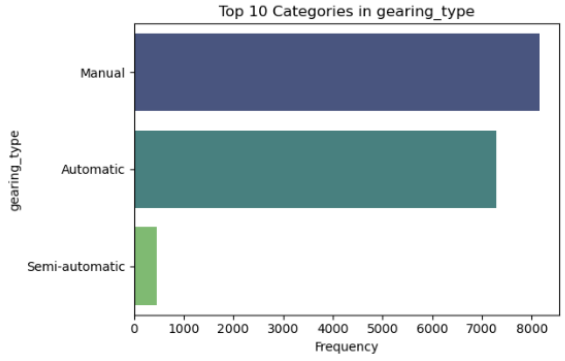

Top 10 Categories in entertainment_media
Bluetooth,Hands-free...
Bluetooth,Hands-free...
Bluetooth,CD player,...
On-board computer
Radio
Bluetooth,Hands-free...
On-board computer,Ra...
Bluetooth,CD player,...
Bluetooth,On-board c...
Bluetooth,CD player,...
entertainment_media
0
250
500
750
1000
1250
1500
1750
Frequency

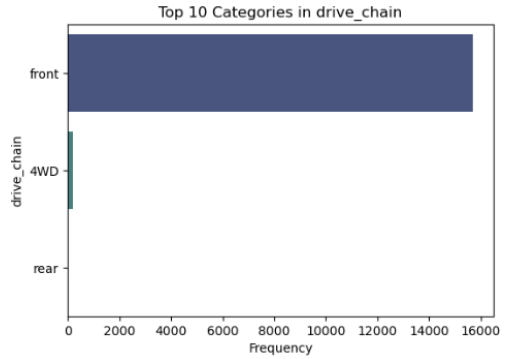

Top 10 Categories in extras
Alloy wheels
Alloy wheels,Touch s...
Roof rack
Alloy wheels,Voice C...
Alloy wheels,Touch s...
Alloy wheels,Roof ra...
Alloy wheels,Sport s...
Alloy wheels,Catalyt...
Alloy wheels,Sport s...
Alloy wheels,Catalyt...
extras
0
1000
2000
3000
4000
5000
6000
Frequency


Top 10 Categories in safety_security
ABS,Central door loc...
ABS,Central door loc...
ABS,Central door loc...
ABS,Central door loc...
ABS,Central door loc...
ABS,Central door loc...
ABS,Central door loc...
ABS,Adaptive headlig...
ABS,Central door loc...
ABS,Central door loc...
safety_security
0
100
200
300
400
500
600
700
Frequency


Top 10 Categories in paint_type
Metallic
Uni/basic
Perl effect
paint_type
0
2000
4000
6000
8000
10000
12000
14000
16000
Frequency


Top 10 Categories in upholstery_type
Cloth
Part/Full Leather
upholstery_type
0
2000
4000
6000
8000
10000
12000
Frequency


Top 10 Categories in gearing_type
Manual
Automatic
Semi-automatic
gearing_type
0
1000
2000
3000
4000
5000
6000
7000
8000
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

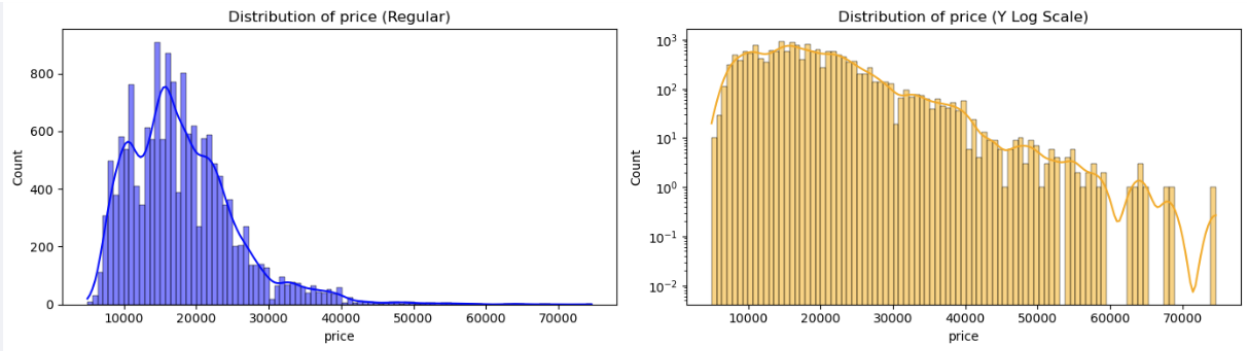
```
--------------------------------------------------------------------------------
Value counts for column 'type':
type
Used              0.697141
New               0.103613
Pre-registered    0.085705
Employee's car    0.063525
Demonstration     0.050016
Name: proportion, dtype: float64

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
```

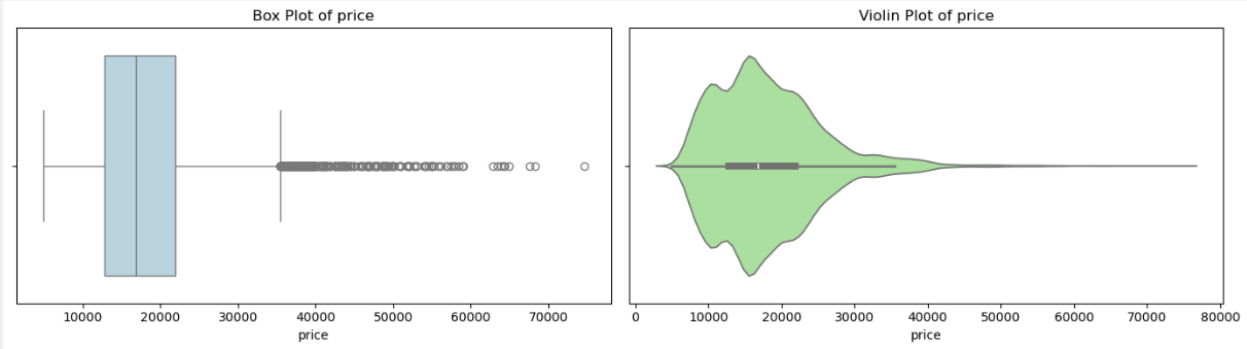
```
--------------------------------------------------------------------------------

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

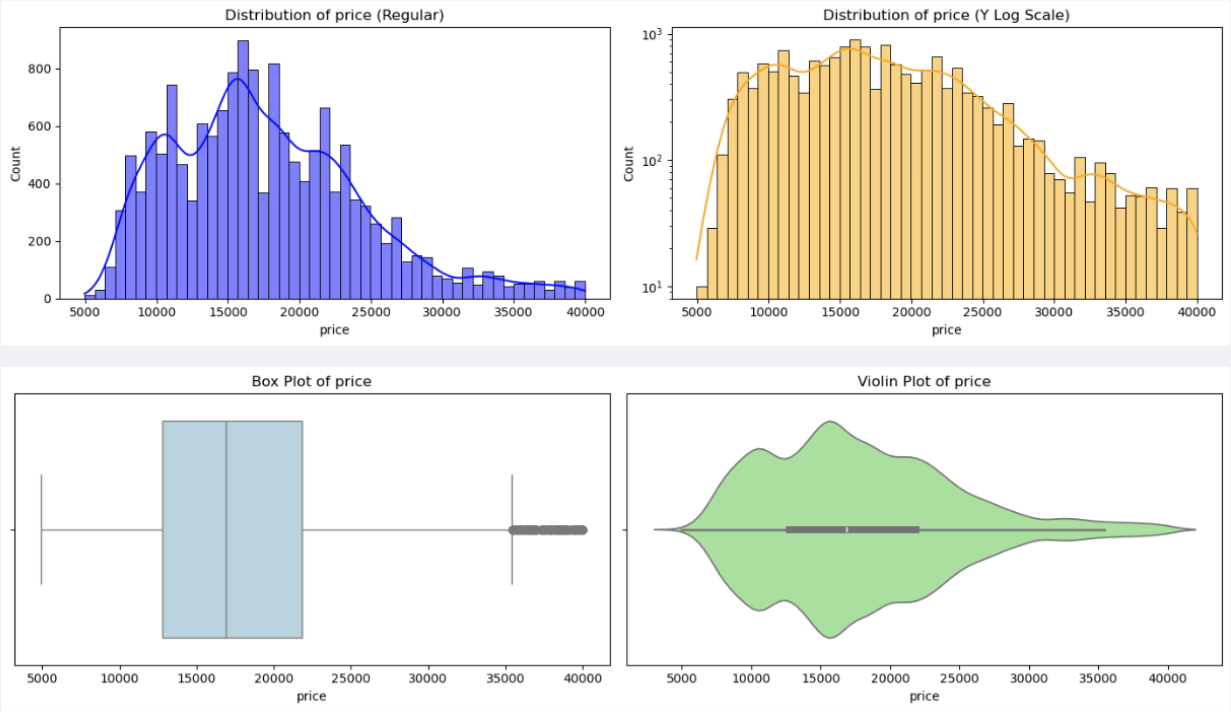
- Plot histogram on ‘Price’ using normal and log y-scale
- Plot box plot and violin plot on ‘Price’
- If target variable is skewed, perform suitable outlier transformation
    - Calculate z-score for each row
    - Identify outliers with absolute(z-score) > threshold
    - Check original record count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





```
Original record count: 15755
Number of outliers in 'price' using Z-score method with threshold 3: 151 (0.96%)
Data shape after removing outliers: (15604, 24)
Records removed: 151

# Check for valid outliers for Audi AX and Renault Espace

Mean price for Audi A3: 20742.805790500977
Mean price for Renault Espace: 27865.379545454547
```

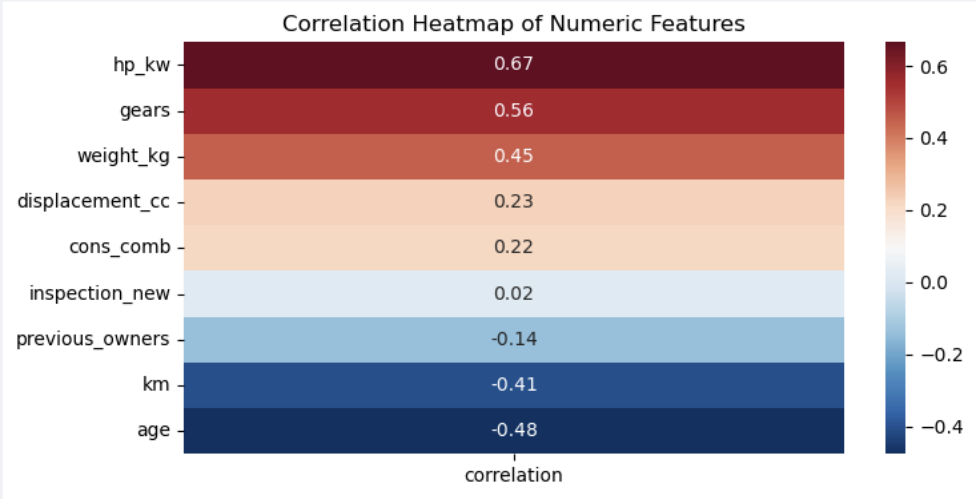
```
Post Outlier Treatment on ['price']:
```



2.2 Correlation Analysis

2.2.1 Correlation Map Between Features and Target Variable

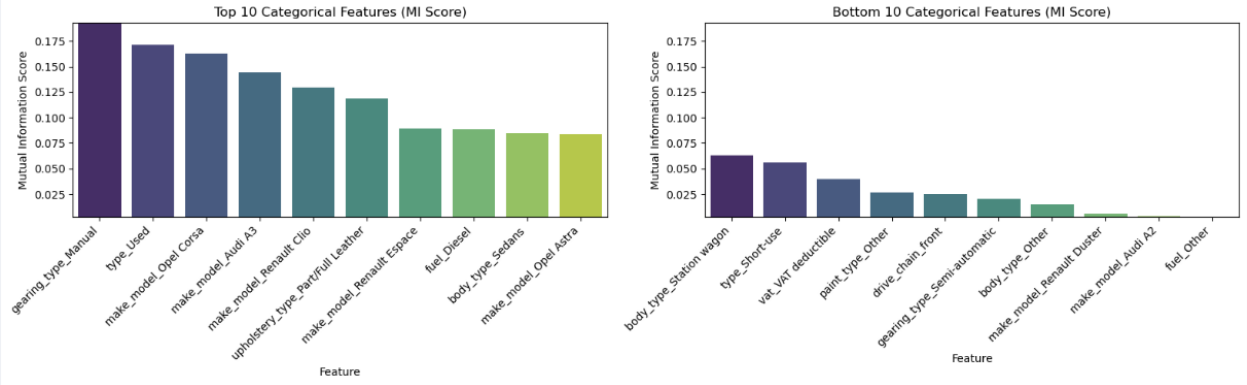
- Separate features and target into X and y
- Find correlation of numeric columns with target variable ‘price’
- Create a data frame with feature name, data type, abs(score), sign
- Plot a heatmap of numeric features correlation
- Display the numeric features with positive or negative high correlation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



	feature	type	score	sign
0	hp_kw	numeric	0.665791	+
1	gears	numeric	0.555941	+
2	age	numeric	0.476724	-
3	weight_kg	numeric	0.448879	+
4	km	numeric	0.407932	-
5	displacement_cc	numeric	0.231158	+
6	cons_comb	numeric	0.217488	+
7	previous_owners	numeric	0.139923	-
8	inspection_new	numeric	0.018830	+

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 y
- Append results into data frame with columns : feature, type = ‘categorical’, score, sign = ‘none’
- Plot bar chart with feature on x-axis and score on y-axis, for Top-k features based on score
- Plot bar chart with feature on x-axis and score on y-axis, for Bottom-k features based on score



MI Score Ranking of Top 10 Categorical Features:

	feature	type	score	sign
0	gearing_type_Manual	categorical	0.192905	None
1	type_Used	categorical	0.171223	None
2	make_model_Opel Corsa	categorical	0.163222	None
3	make_model_Audi A3	categorical	0.144701	None
4	make_model_Renault Clio	categorical	0.129497	None
5	upholstery_type_Part/Full Leather	categorical	0.118510	None
6	make_model_Renault Espace	categorical	0.089365	None
7	fuel_Diesel	categorical	0.088448	None
8	body_type_Sedans	categorical	0.084705	None
9	make_model_Opel Astra	categorical	0.084110	None

MI Score Ranking of Bottom 10 Categorical Features:

	feature	type	score	sign
12	body_type_Station wagon	categorical	0.062802	None
13	type_Short-use	categorical	0.055795	None
14	vat_VAT deductible	categorical	0.039553	None
15	paint_type_Other	categorical	0.026313	None
16	drive_chain_front	categorical	0.025086	None
17	gearing_type_Semi-automatic	categorical	0.020173	None
18	body_type_Other	categorical	0.014975	None
19	make_model_Renault Duster	categorical	0.005422	None
20	make_model_Audi A2	categorical	0.002962	None
21	fuel_Other	categorical	0.002349	None

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

	feature	type	score	sign
0	hp_kw	numeric	0.665791	+
1	gears	numeric	0.555941	+
2	age	numeric	0.476724	-
3	weight_kg	numeric	0.448879	+
4	km	numeric	0.407932	-
5	displacement_cc	numeric	0.231158	+
6	cons_comb	numeric	0.217488	+
7	gearing_type_Manual	categorical	0.192905	None
8	type_Used	categorical	0.171223	None
9	make_model_Opel Corsa	categorical	0.163222	None
10	make_model_Audi A3	categorical	0.144701	None
11	previous_owners	numeric	0.139923	-
12	make_model_Renault Clio	categorical	0.129497	None
13	upholstery_type_Part/Full Leather	categorical	0.118510	None
14	make_model_Renault Espace	categorical	0.089365	None
15	fuel_Diesel	categorical	0.088448	None
16	body_type_Sedans	categorical	0.084705	None
17	make_model_Opel Astra	categorical	0.084110	None
18	make_model_Opel Insignia	categorical	0.078558	None
19	body_type_Van	categorical	0.065316	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.

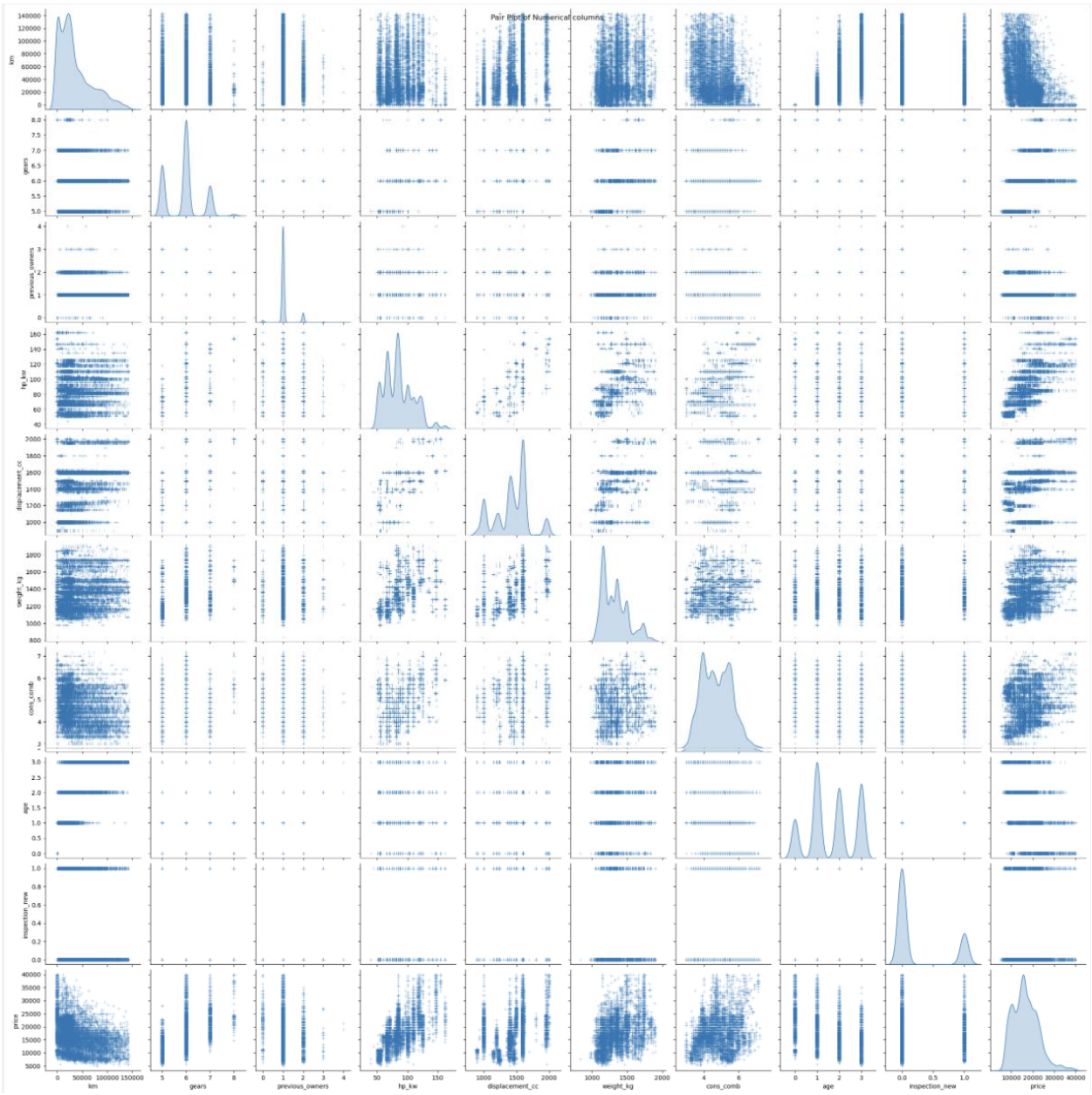
- **Numerical features** are ranked by absolute Pearson correlation with the target.
- **Categorical features** are ranked by mutual information (MI) with the target.
- This combined ranking helps prioritize which features to focus on for modeling and interpretation.

Rank	Feature	Type	Score	Sign	Inference
1	hp_kw	Numeric	0.666	+	Strongest driver of price. More engine power → higher price.
2	gears	Numeric	0.556	+	More gears (modern transmissions) → generally higher prices.
3	age	Numeric	0.477	–	Older cars lose value. Strong negative effect.
4	weight_kg	Numeric	0.449	+	Heavier cars (luxury/SUVs) are costlier.
5	km	Numeric	0.408	–	Higher mileage reduces price, though less than age.
6	displacement_cc	Numeric	0.231	+	Bigger engines → higher prices, though weaker than horsepower.

7	cons_comb	Numeric	0.217	+	Higher consumption → often premium/performance cars.
8	gearing_type_Manual	Categorical	0.193	–	Manual cars usually cheaper than automatics (signal of lower price).
9	type_Used	Categorical	0.171	–	Used cars priced lower than new ones.
10	make_model_Opel Corsa	Categorical	0.163	–	Budget model → pulls price down.
11	make_model_Audi A3	Categorical	0.145	+	Premium model → lifts price.
12	previous_owners	Numeric	0.140	–	More owners reduce resale value.
13	make_model_Renault Clio	Categorical	0.129	–	Mass-market model, generally lower priced.
14	upholstery_type_Part/Full Leather	Categorical	0.119	+	Premium interiors increase price.
15	make_model_Renault Espace	Categorical	0.089	–	Family van, not premium → lower prices.
16	fuel_Diesel	Categorical	0.088	±	Diesel cars affect price; historically valuable in EU, now mixed.
17	body_type_Sedans	Categorical	0.085	±	Sedans have moderate premium depending on market.
18	make_model_Opel Astra	Categorical	0.084	–	Mid-segment Opel → relatively lower priced.
19	make_model_Opel Insignia	Categorical	0.079	–	Slightly higher than Astra but still budget vs luxury brands.
20	body_type_Van	Categorical	0.065	–	Vans typically lower resale vs SUVs/sedans.

2.2.2.2 Pair Plot for Correlation Analysis

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

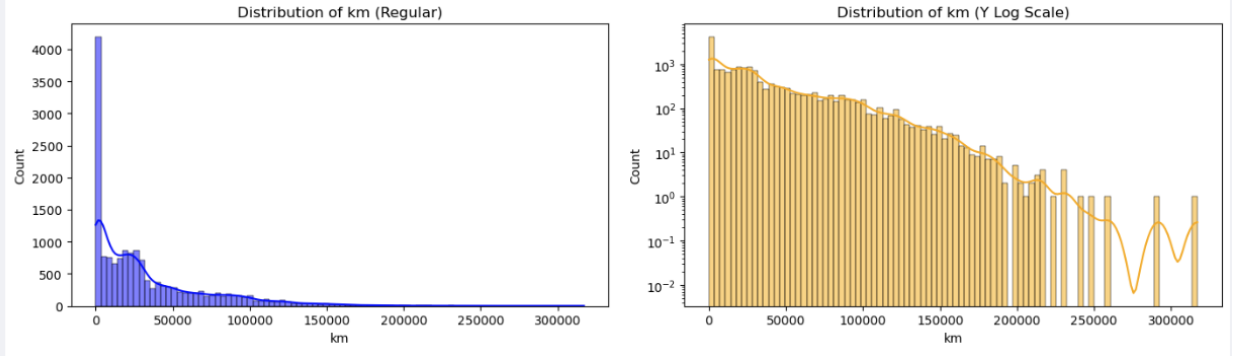


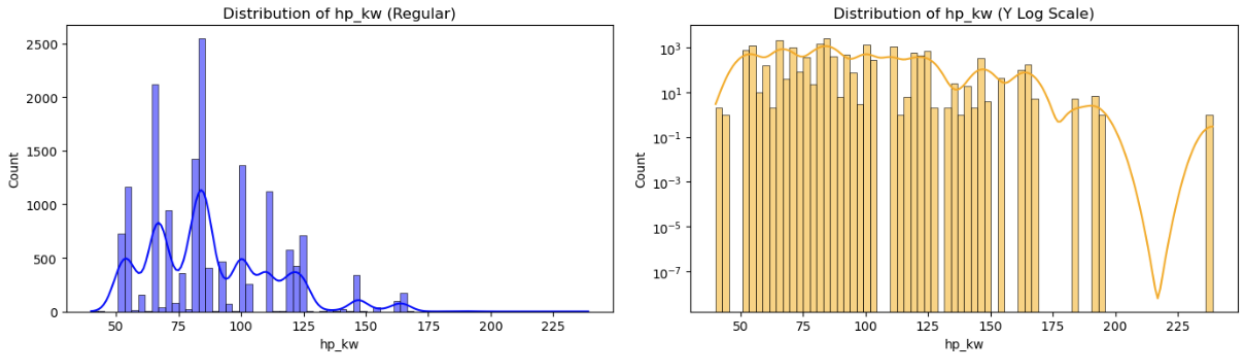
The pair plot confirms the correlations as identified earlier between numeric categorical variables and target ‘price’.

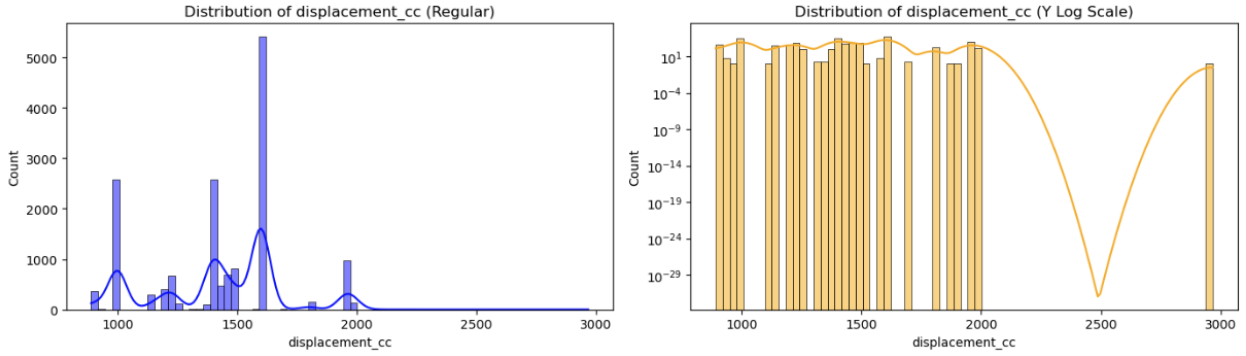
2.3 Outlier Analysis

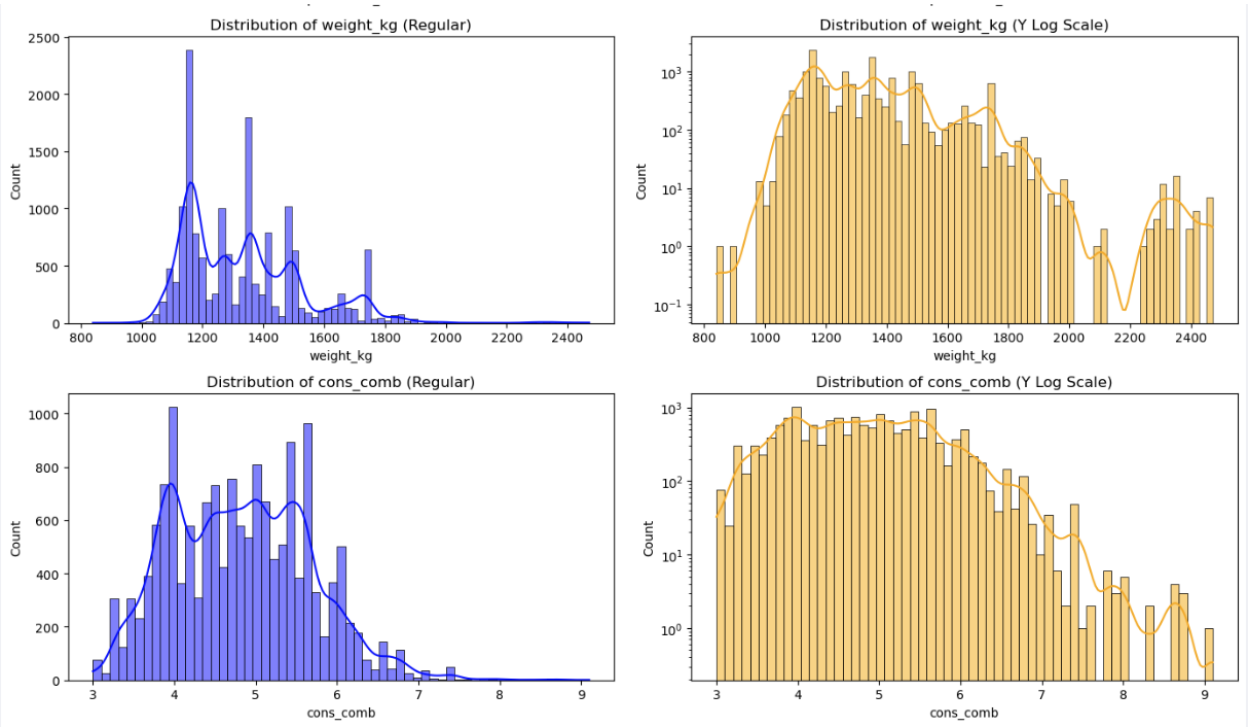
- Identify outliers in the numeric categorical variables
- Plot histogram with regular and log y-scale
- Plot box plot and violin plot

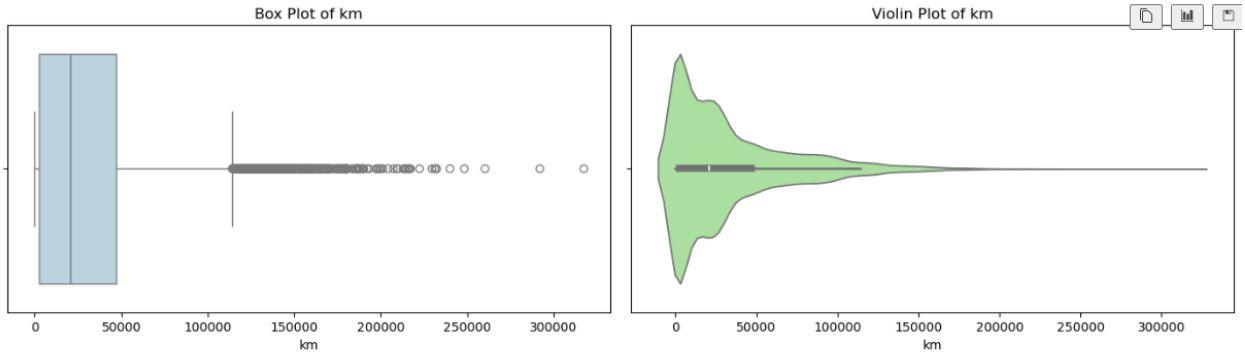
```
Pre Outlier Treatment on ['km', 'hp_kw', 'displacement_cc', 'weight_kg', 'cons_comb']:
```

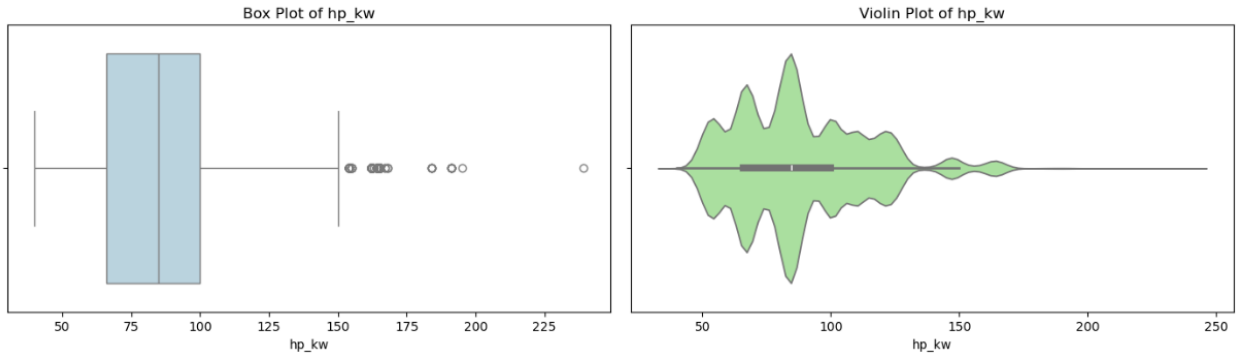


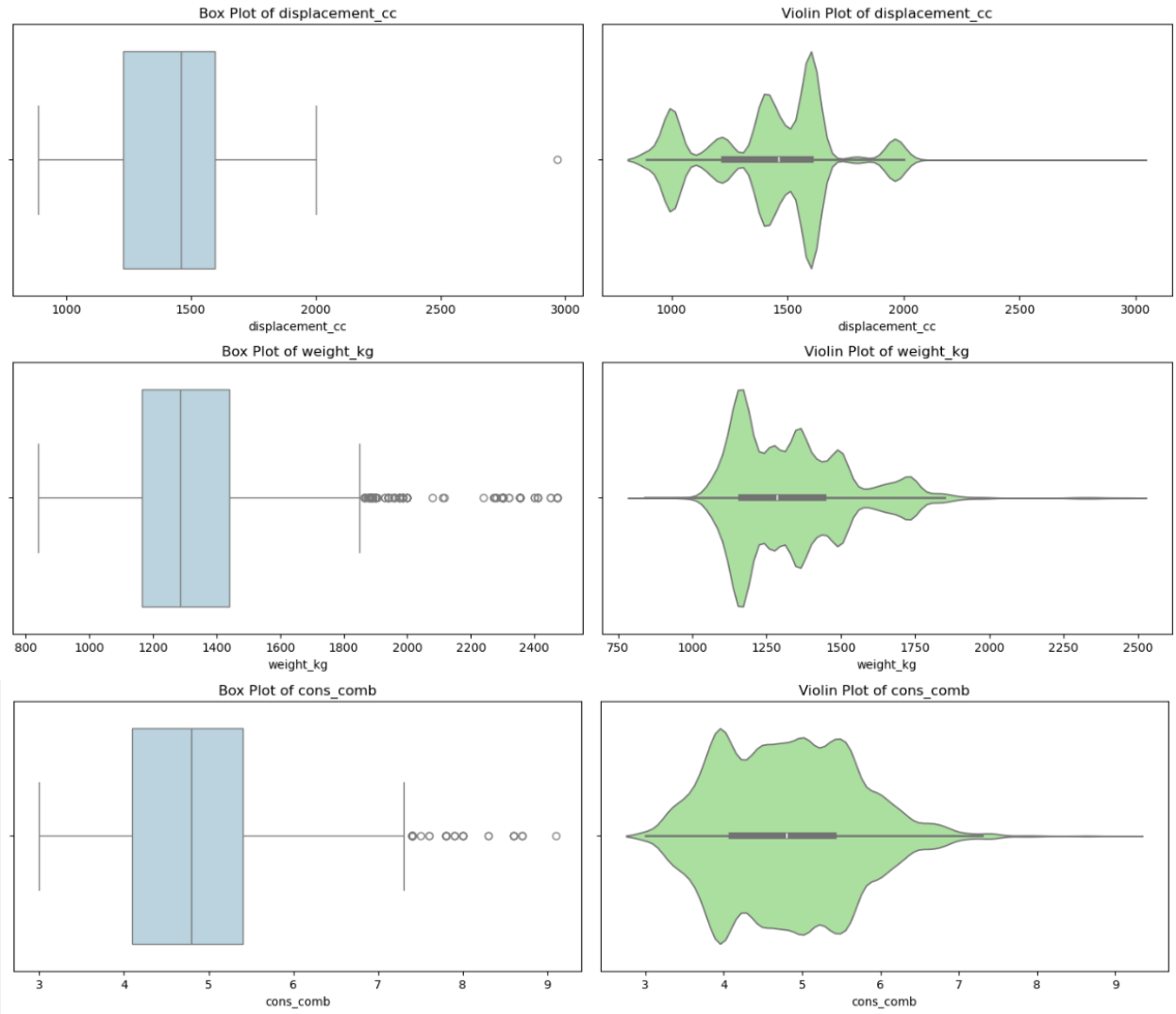





Distribution of weight_kg (Regular)
Count
weight_kg
Distribution of weight_kg (Y Log Scale)
Count
weight_kg
Distribution of cons_comb (Regular)
Count
cons_comb
Distribution of cons_comb (Y Log Scale)
Count
cons_comb


Box Plot of km
km
Violin Plot of km
km


Box Plot of hp_kw
hp_kw
Violin Plot of hp_kw
hp_kw

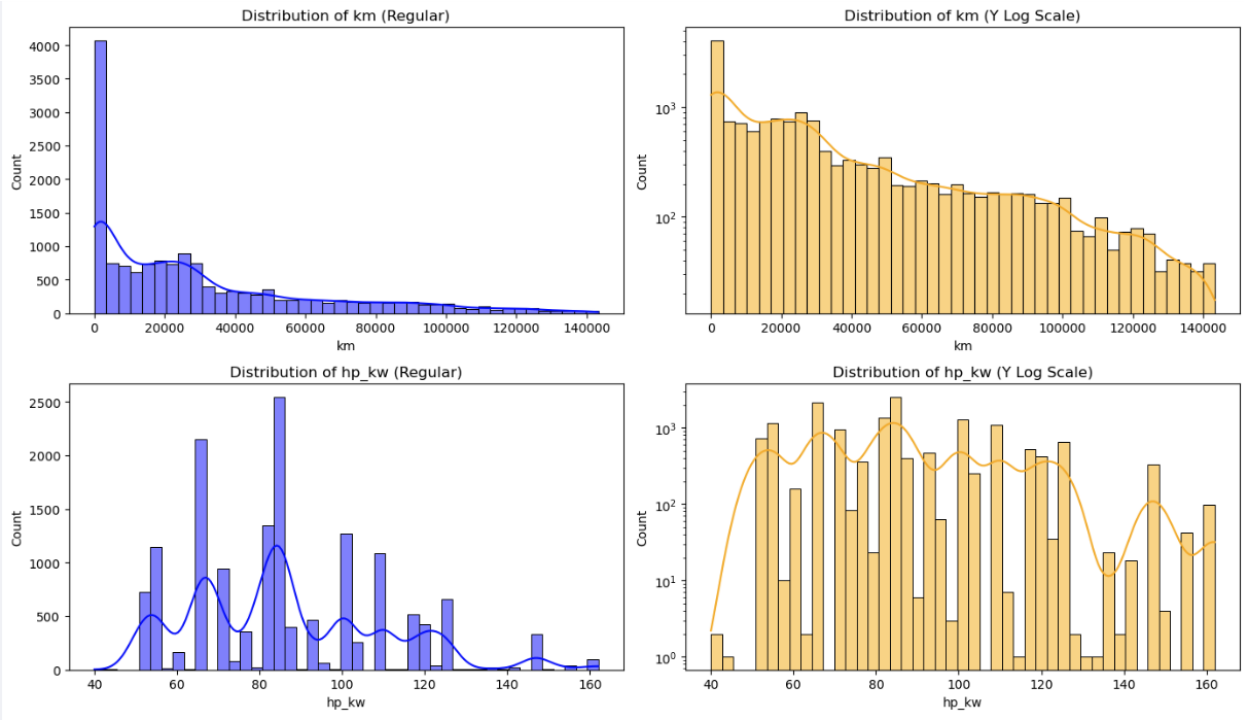


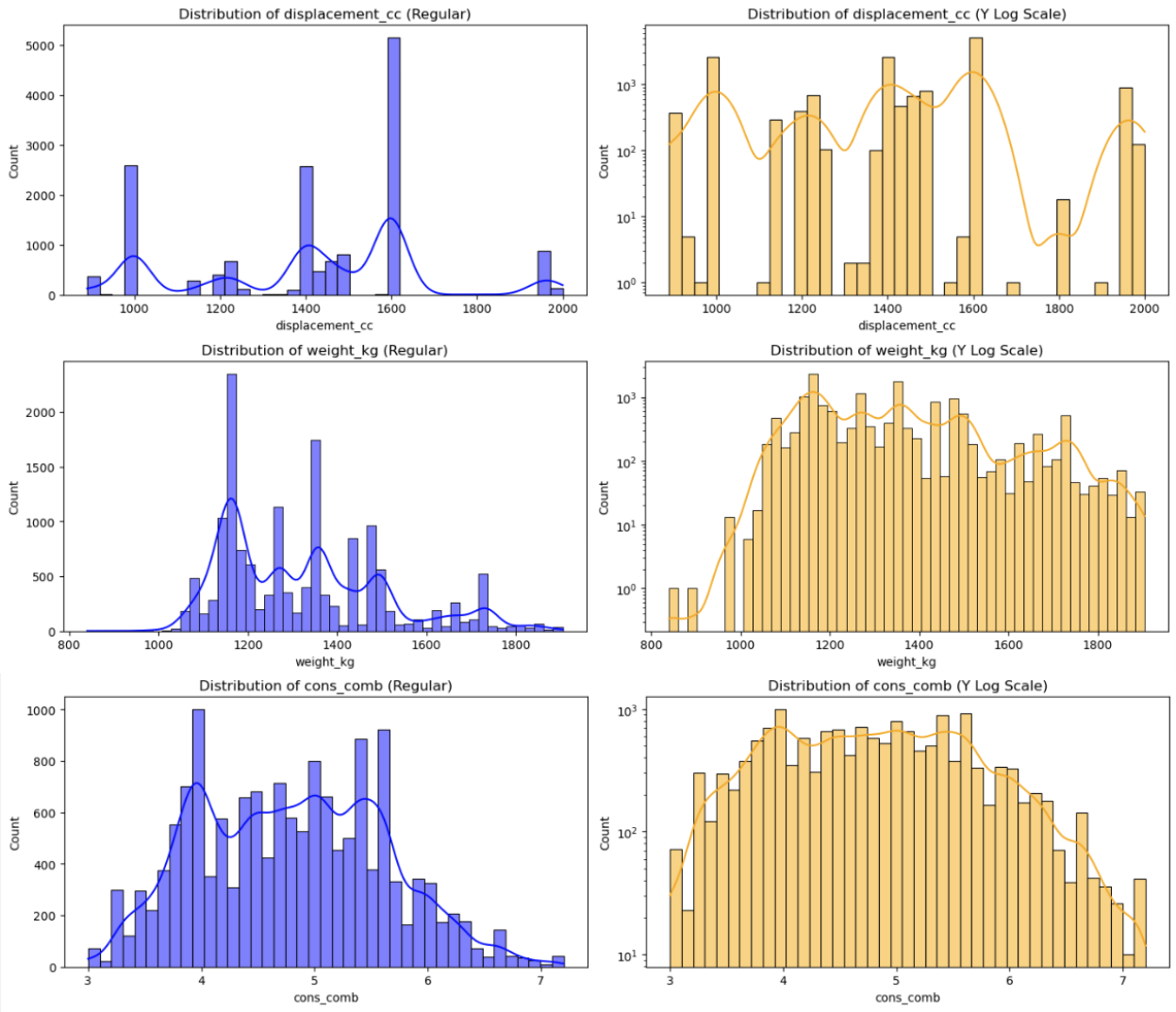
2.3.1 Handle Outliers in Numeric Categorical Variables

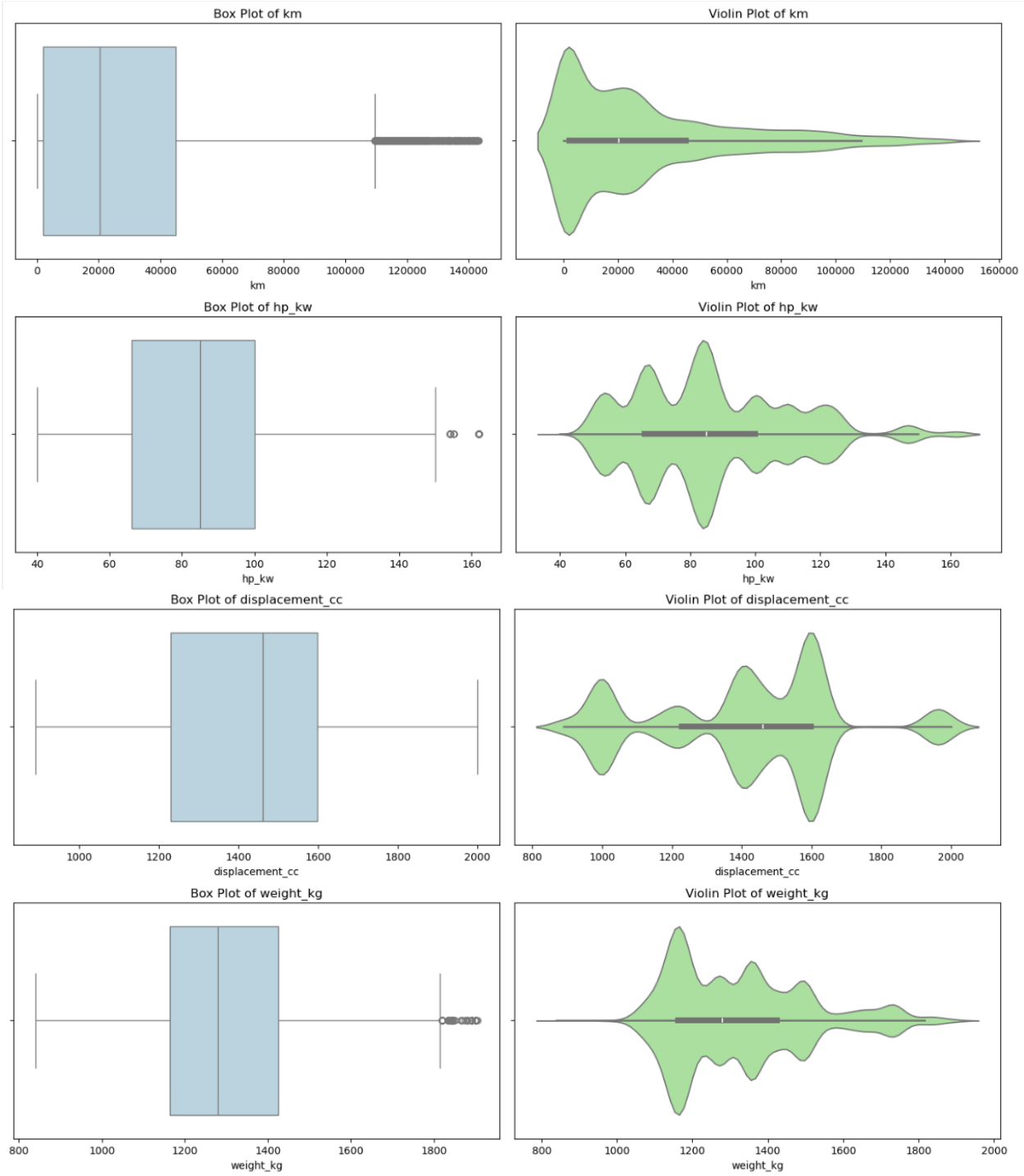
- Use z-score functioned defined earlier to identify outliers using threshold between 2 and 3
- Remove outliers, which are beyond 3 sigma deviation
- Plot histograms, box and violin charts to validate outcomes

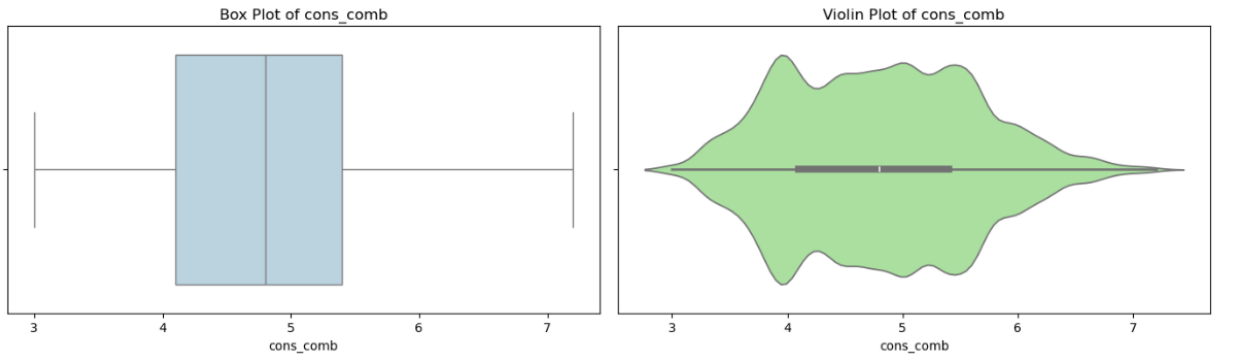
```
--------------------------------------------------------------------------------
Original record count: 15755
Number of outliers in 'km' using Z-score method with threshold 3: 260 (1.65%)
Data shape after removing outliers: (15495, 24)
Records removed: 260
--------------------------------------------------------------------------------
Original record count: 15495
Number of outliers in 'hp_kw' using Z-score method with threshold 3: 189 (1.22%)
Data shape after removing outliers: (15306, 24)
Records removed: 189
--------------------------------------------------------------------------------
Original record count: 15306
```

```
Number of outliers in 'displacement_cc' using Z-score method with threshold 3: 0 (0.00%)
Data shape after removing outliers: (15306, 24)
Records removed: 0
-------------------------------------------------------------------------------
Original record count: 15306
Number of outliers in 'weight_kg' using Z-score method with threshold 3: 83 (0.54%)
Data shape after removing outliers: (15223, 24)
Records removed: 83
-------------------------------------------------------------------------------
Original record count: 15223
Number of outliers in 'cons_comb' using Z-score method with threshold 3: 17 (0.11%)
Data shape after removing outliers: (15206, 24)
Records removed: 17
```




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


Box Plot of km
0
20000
40000
60000
80000
100000
120000
140000
km
Violin Plot of km
0
20000
40000
60000
80000
100000
120000
140000
160000
km
Box Plot of hp_kw
40
60
80
100
120
140
160
hp_kw
Violin Plot of hp_kw
40
60
80
100
120
140
160
hp_kw
Box Plot of displacement_cc
1000
1200
1400
1600
1800
2000
displacement_cc
Violin Plot of displacement_cc
800
1000
1200
1400
1600
1800
2000
displacement_cc
Box Plot of weight_kg
800
1000
1200
1400
1600
1800
weight_kg
Violin Plot of weight_kg
800
1000
1200
1400
1600
1800
2000
weight_kg



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 4203 cars with zero age which is valid for new cars.
- There are 1797 cars with zero age and high km.
- There are 1494 cars with zero age and type <> new.
- There are 862 cars with zero age, high km and type <> new.
- These records are likely data quality issues and will be removed.

Data Quality Counts Post Cleanup - age
- There are 1774 cars with zero age which is valid for new cars.
- There are 0 cars with zero age and high km.
- There are 0 cars with zero age and type <> new.
- There are 0 cars with zero age, high km and type <> new.
- These records are likely data quality issues and will be removed.
---------------------------------------------------------------------------

Data Quality Checks Driven Cleanup - km
- There are 19 cars with zero km which is valid for new cars.
- There are 0 cars with zero km and type <> new.
- There are 0 cars with zero km, age <> 0.
- These look like brand new cars, example the ones in showroom without usage, hence
will be retained.

---------------------------------------------------------------------------
Checking for realistic value ranges for ['Audi A1', 'Audi A2', 'Audi A3', 'Opel
Astra', 'Opel Corsa','Opel Insignia', 'Renault Clio', 'Renault Espace']
- Car weight should be positive
- Engine displacement should be positive
- Car weight should be between 600 and 3500 kg
- Displacement should be between 890 and 3000 cc
```

```
- Engine power (hp_kw) should be between 40 and 300 kW
- Combined fuel consumption (cons_comb) should be between 3 and 15 L/100km
- Car age should be between 0 and 30 years
- Car mileage (km) should be between 0 and 400,000 km
--------------------------------------------------------------------------------

Post Outlier Treatment Summary Statistics (Min, Max, Missing, Negative, Zero counts):
- weight_kg: min=840.0, max=1905.0, missing=0, negative=0, zero=0
- displacement_cc: min=898.0, max=2000.0, missing=0, negative=0, zero=0
- hp_kw: min=40.0, max=162.0, missing=0, negative=0, zero=0
- cons_comb: min=3.0, max=7.2, missing=0, negative=0, zero=0
- age: min=0.0, max=3.0, missing=0, negative=0, zero=1774
- km: min=0.0, max=143258.0, missing=0, negative=0, zero=19
--------------------------------------------------------------------------------

Post Z-score Outlier Treatment - Records above 99th percentile:
- Weight (kg) above 99th percentile: 128
- Displacement (cc) above 99th percentile: 80
- HP (kW) above 99th percentile: 116
- Fuel consumption (cons_comb) above 99th percentile: 98
--------------------------------------------------------------------------------
Post Outlier Treatment and Data Validation:
- All values of weight_kg, displacement_cc, hp_kw, cons_comb, age, km are within
realistic ranges
- No missing or negative values in key columns weight_kg, displacement_cc, hp_kw,
cons_comb, age, km
- There are few records above 99th percentile in Weight, Displacement, HP, fuel
consumption which are valid high-end cars
```

2.4 Feature Engineering

2.4.1 Fix Redundant Columns and Create New Ones

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 12777 entries, 0 to 15913
Data columns (total 35 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   price                               12777 non-null  int64
 1   km                                  12777 non-null  float64
 2   gears                               12777 non-null  float64
 3   comfort_convenience                 12777 non-null  object
 4   entertainment_media                 12777 non-null  object
 5   extras                              12777 non-null  object
 6   safety_security                     12777 non-null  object
 7   age                                 12777 non-null  float64
 8   previous_owners                     12777 non-null  float64
 9   hp_kw                               12777 non-null  float64
 10  inspection_new                      12777 non-null  int64
 11  displacement_cc                     12777 non-null  float64
 12  weight_kg                           12777 non-null  float64
 13  cons_comb                           12777 non-null  float64
```

```
 14  make_model_Audi A2                 12777 non-null  int64
 15  make_model_Audi A3                 12777 non-null  int64
 16  make_model_Opel Astra              12777 non-null  int64
 17  make_model_Opel Corsa              12777 non-null  int64
 18  make_model_Opel Insignia           12777 non-null  int64
 19  make_model_Renault Clio            12777 non-null  int64
...
 33  gearing_type_Semi-automatic        12777 non-null  int64
 34  drive_chain_front                  12777 non-null  int64
dtypes: float64(8), int64(23), object(4)
```

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: 26 features retained out of 38 total
entertainment_media: 9 features retained out of 10 total
extras: 8 features retained out of 17 total
safety_security: 27 features retained out of 29 total

List of commonly used features retained in the columns:
{'comfort_convenience': ['Air conditioning',
                         'Armrest',
                         'Automatic climate control',
                         'Cruise control',
                         'Electrical side mirrors',
                         'Hill Holder',
                         'Leather steering wheel',
                         'Light sensor',
                         'Multi-function steering wheel',
                         'Navigation system',
                         'Park Distance Control',
                         'Parking assist system sensors rear',
                         'Power windows',
                         'Rain sensor',
                         'Seat heating',
                         'Start-stop system',
                         'Lumbar support',
                         'Tinted windows',
                         'Parking assist system sensors front',
                         'Split rear seats',
                         'Keyless central door lock',
                         'Electrically heated windshield',
                         'Parking assist system camera',
                         'Electrically adjustable seats',
                         'Electric tailgate',
                         'Heated steering wheel'],
 'entertainment_media': ['Bluetooth',
                         'Hands-free equipment',
                         'On-board computer',
                         'Radio',
                         'Sound system',
                         'MP3',
```

```
                        'CD player',
                        'USB',
                        'Digital radio'],
 'extras': ['Alloy wheels',
            'Catalytic Converter',
            'Voice Control',
            'Sport seats',
            'Sport suspension',
            'Sport package',
            'Touch screen',
            'Roof rack'],
 'safety_security': ['ABS',
                     'Central door lock',
                     'Daytime running lights',
                     'Driver-side airbag',
                     'Electronic stability control',
                     'Fog lights',
                     'Immobilizer',
                     'Isofix',
                     'Passenger-side airbag',
                     'Power steering',
                     'Side airbag',
                     'Tire pressure monitoring system',
                     'Traction control',
                     'Xenon headlights',
                     'Central door lock with remote control',
                     'Head airbag',
                     'Alarm system',
                     'Emergency system',
                     'LED Headlights',
                     'Adaptive headlights',
                     'LED Daytime Running Lights',
                     'Rear airbag',
                     'Emergency brake assistant',
                     'Adaptive Cruise Control',
                     'Traffic sign recognition',
                     'Lane departure warning system',
                     'Blind spot monitor']}
```

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: (12777, 101)

<class 'pandas.core.frame.DataFrame'>
Index: 12777 entries, 0 to 15913
Columns: 101 entries, price to safety_security_Blind spot monitor
dtypes: float64(8), int64(93)
memory usage: 9.9 MB
```

2.4.4 Split Data into Training and Test

- Separate dependent and independent variables into X and y

- Create X_train, X_test, y_train, y_test dataframes using test size 0.2 and random state 42 (for consistency across runs)

```
Shape of X_train: (10221, 100), y_train: (10221,), X_test: (2556, 100), y_test:
(2556,)
```

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 y_train using the fitted scalar

```
Shape of X_train_scaled, X_test_scaled : (10103, 99), (2526, 99)
```

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_pred, y_test_pred using the fitted linear regression model
- Create a lr_coeff dataframe with features, coefficients, absolute(coefficients), sign
- Display top-N coefficients using absolute values
- Analyze model scores for both Train and Test data –
    - R2, MSE, RMSE, MAE, MAPE using y_actual and y_predicted values
    - Variance, Standard Deviation, Mean using y_actual values

```
--------------------------------------------------------------------------------
Top 10 Coefficients with sign
--------------------------------------------------------------------------------
```

	feature	coef	coef_abs	sign
12	make_model_Opel Corsa	-1873.558987	1873.558987	-
4	hp_kw	1786.336836	1786.336836	+
14	make_model_Renault Clio	-1730.773717	1730.773717	-
11	make_model_Opel Astra	-1268.098820	1268.098820	-
0	km	-1215.942717	1215.942717	-
2	age	-1195.504940	1195.504940	-
27	gearing_type_Manual	-807.496099	807.496099	-
15	make_model_Renault Espace	784.069910	784.069910	+
22	type_Used	-710.314486	710.314486	-
10	make_model_Audi A3	676.506593	676.506593	+

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

	feature	coef	coef_abs	sign
3	previous_owners	-1.856752e+01	1.856752e+01	-
57	entertainment_media_Hands-free equipment	1.802845e+01	1.802845e+01	+
38	comfort_convenience_Multi-function steering wheel	1.383052e+01	1.383052e+01	+
89	safety_security_Alarm system	-1.140505e+01	1.140505e+01	-
58	entertainment_media_On-board computer	-8.433080e+00	8.433080e+00	-
42	comfort_convenience_Power windows	-7.806964e+00	7.806964e+00	-
53	comfort_convenience_Electrically adjustable seats	4.584537e+00	4.584537e+00	+
8	cons_comb	4.470476e+00	4.470476e+00	+
51	comfort_convenience_Electrically heated windsh...	4.345203e+00	4.345203e+00	+
9	make_model_Audi A2	-4.547474e-13	4.547474e-13	-

```
--------------------------------------------------------------------------------
Analyze Model Performance - Baseline Linear Regression
--------------------------------------------------------------------------------
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1916.395641	5964.193905	3.672572e+06	3.557161e+07	1339.295942	16687.750807	0.088136	0.896745
1	Test	1927.425144	5953.809415	3.714968e+06	3.544785e+07	1326.169542	16695.843505	0.086239	0.895158

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.

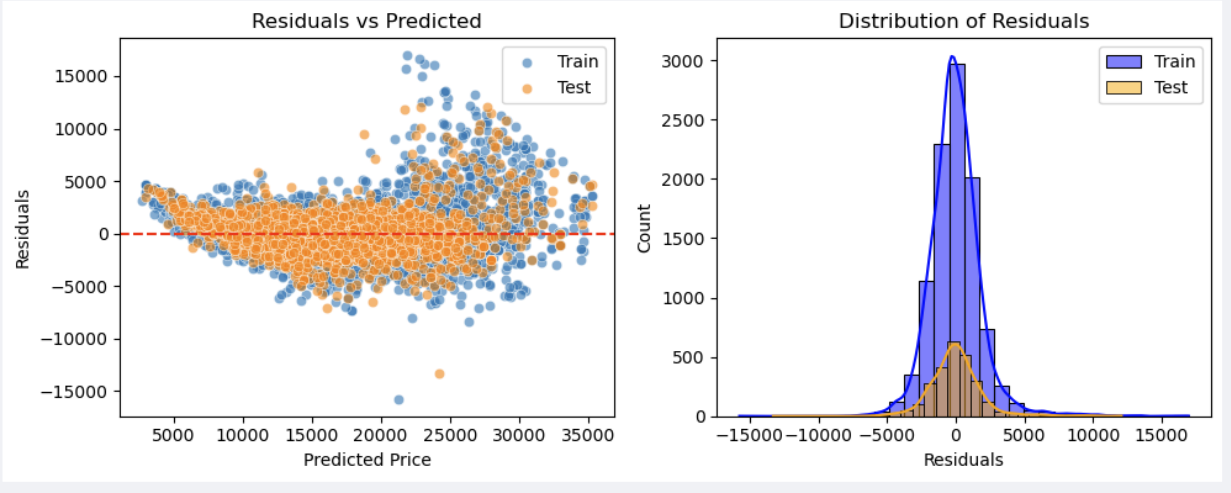
3.1.1.2 Summary:

- 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

- 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
```



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	intercept	898.164917
1	make_model_Renault Espace	9.197915
2	hp_kw	7.419921
3	fuel_Diesel	7.367117
4	make_model_Opel Insignia	7.200831
5	displacement_cc	6.629169
6	body_type_Van	6.370844
7	make_model_Opel Corsa	5.516246
8	weight_kg	5.094156
9	comfort_convenience_Parking assist system sens...	4.976697
10	body_type_Station wagon	4.667394
11	cons_comb	4.528100
12	make_model_Opel Astra	4.451727
13	comfort_convenience_Park Distance Control	4.179142
14	age	3.959336
15	make_model_Audi A3	3.885996
16	make_model_Renault Clio	3.836322
17	gears	3.171619
18	type_Used	3.069647
19	comfort_convenience_Heated steering wheel	2.910976

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
intercept	898	Ignore. Intercept often has artificially high VIF and doesn't affect feature decisions.
make_model_Renault Espace	9.19	High. Likely overlaps with body_type (Van) and maybe weight_kg.
hp_kw	7.41	High. Horsepower is highly correlated with displacement_cc and weight_kg.
fuel_Diesel	7.36	High. Fuel type interacts with body_type and consumption (cons_comb).
make_model_Opel Insignia	7.20	High. Strongly collinear with other Opel models + body types.
displacement_cc	6.62	Multicollinear with hp_kw (both capture engine size/power).
body_type_Van	6.37	Overlaps with Renault Espace (most Espace are vans) → redundancy.
make_model_Opel Corsa	5.51	Moderate collinearity, probably with other Opel models.
weight_kg	5.10	Correlated with hp_kw and displacement_cc. Bigger engines = heavier cars.
comfort_convenience_Parking assist…	4.98	Moderate. Likely overlaps with other comfort features (Park Distance Control).

body_type_Station wagon	4.66	Overlaps with specific models (Opel Astra wagon, Renault Espace).
cons_comb	4.52	Correlated with engine size (displacement_cc, hp_kw).
make_model_Opel Astra	4.45	Model dummies always show collinearity with body_type & brand group.
comfort_convenience_Park Distance Control	4.18	Redundant with Parking assist system sensors.
age	3.96	Low–moderate. Correlates slightly with km (which you probably dropped earlier due to multicollinearity).
make_model_Audi A3	3.89	Fine, moderate. Captures brand-specific pricing.
make_model_Renault Clio	3.83	Acceptable. Still collinear with Renault brand dummies.
gears	3.17	Acceptable. Correlates with hp_kw (higher performance cars → more gears).
type_Used	3.06	Acceptable. Correlates slightly with age and km.
comfort_convenience_Rain sensor	2.91	Low, no issue.

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

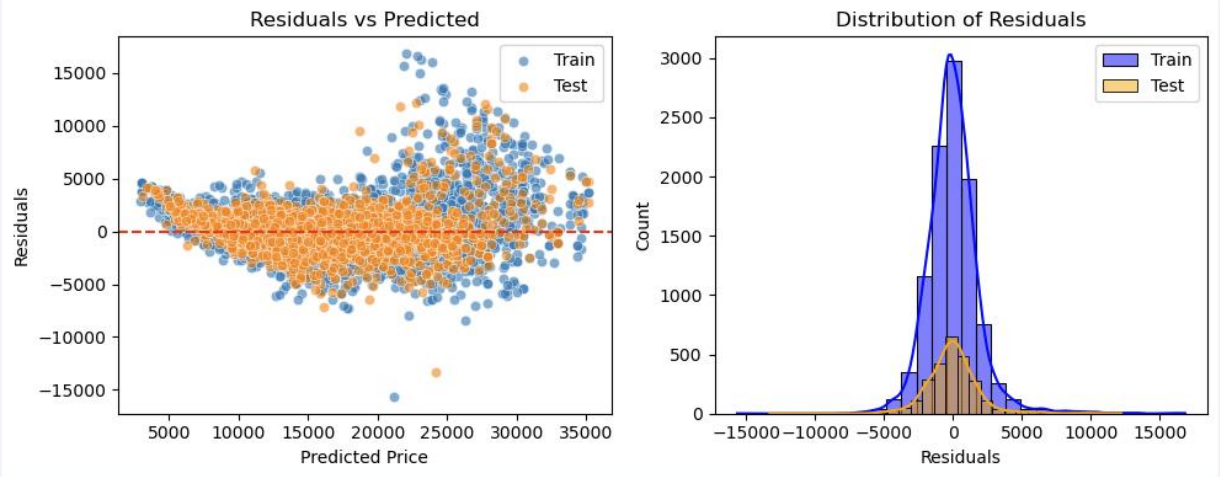
3.2.2.2 Analyze Model Performance & Identify Chosen Alpha

- 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

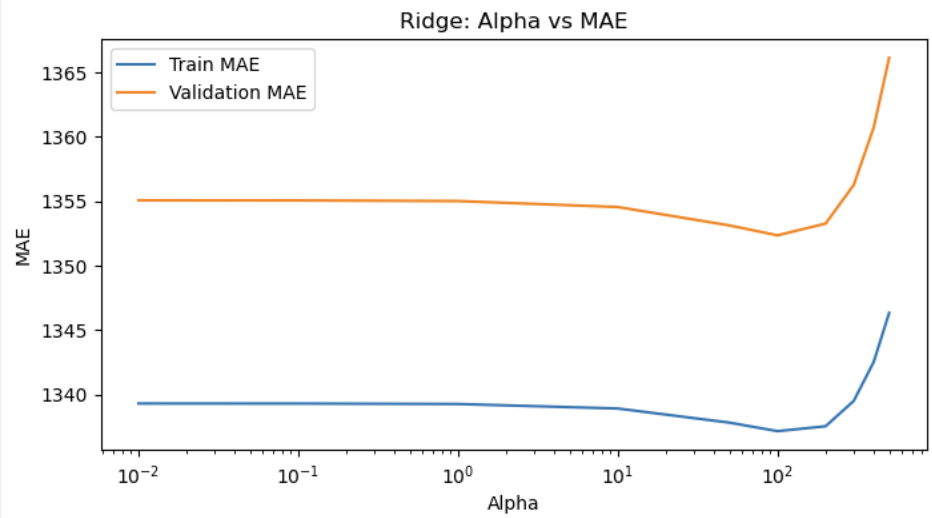
	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1918.045885	5964.193905	3.678900e+06	3.557161e+07	1337.148364	16687.750807	0.087747	0.896568
1	Test	1927.496661	5953.809415	3.715243e+06	3.544785e+07	1323.690419	16695.843505	0.085888	0.895150

RidgeCV Model – Plot of Residuals vs Predicted



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



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): 100.0
--------------------------------------------------------------------------------
```

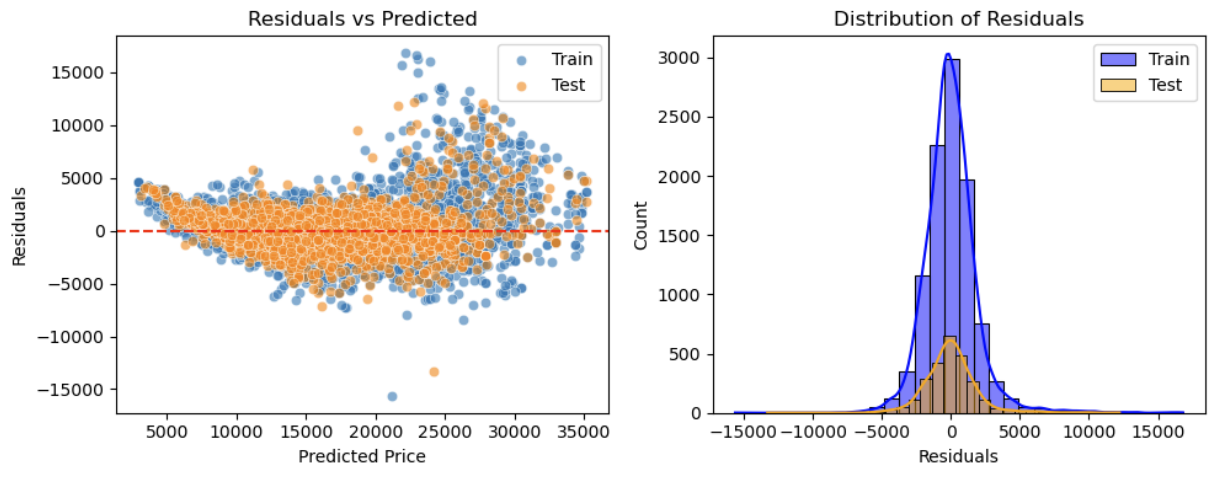
3.2.3 Fine Tune Alpha, Analyze Performance, Error-Alpha Chart, Coefficients

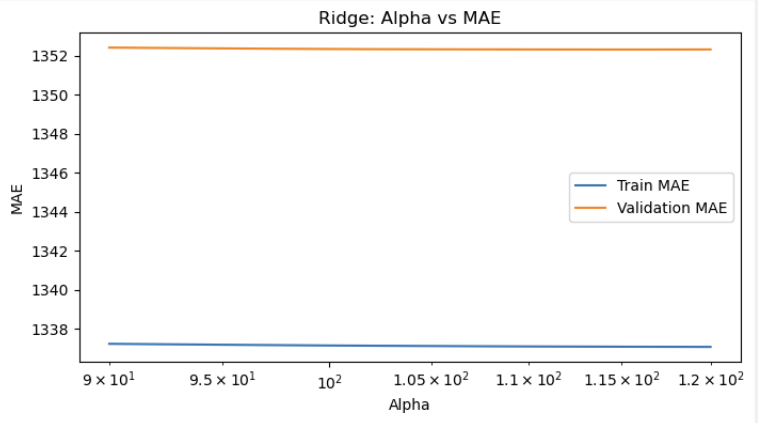
- We will narrow down the alpha range around the chosen alpha value from 3.2.2.4
- narrow_alpha_range = np.*linspace*(90, 120, 30) # 30 values between 90 and 120
- Run the resusable function ridgecv model using the narrow alpha range
- Analyze the model performance on train and test data
    - Plot residuals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

```
Final RidgeCV Model – Analyze Model Performance
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1918.382376	5964.193905	3.680191e+06	3.557161e+07	1337.096922	16687.750807	0.087718	0.896531
1	Test	1927.690605	5953.809415	3.715991e+06	3.544785e+07	1323.564053	16695.843505	0.085860	0.895129

```
Final RidgeCV Model – Plot of Residuals vs Predicted
Best RidgeCV Model Score (Negative MAE): 1323.5640531766749
Best RidgeCV Model Alpha: 110.6896551724138
```





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

```
--------------------------------------------------------------------------------
Final Ridge Model Coefficients with best score alpha=110.6896551724138:
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
Top 10 Coefficients (based on absolute value) with sign
--------------------------------------------------------------------------------
```

	feature	coef	coef_abs	sign
12	make_model_Opel Corsa	-1767.951299	1767.951299	-
4	hp_kw	1702.835831	1702.835831	+
14	make_model_Renault Clio	-1609.584789	1609.584789	-
0	km	-1204.063414	1204.063414	-
2	age	-1191.240143	1191.240143	-
11	make_model_Opel Astra	-1155.893700	1155.893700	-
15	make_model_Renault Espace	804.014732	804.014732	+
27	gearing_type_Manual	-799.542647	799.542647	-
22	type_Used	-709.050511	709.050511	-
10	make_model_Audi A3	701.383551	701.383551	+

```
--------------------------------------------------------------------------------
Bottom 10 Coefficients (based on absolute value) with sign
--------------------------------------------------------------------------------
```

	feature	coef	coef_abs	sign
44	comfort_convenience_Seat heating	-18.511696	18.511696	-
92	safety_security_Adaptive headlights	-18.509674	18.509674	-
38	comfort_convenience_Multi-function steering wheel	15.906676	15.906676	+
57	entertainment_media_Hands-free equipment	11.115811	11.115811	+
89	safety_security_Alarm system	-10.672150	10.672150	-
67	extras_Voice Control	-10.198033	10.198033	-
42	comfort_convenience_Power windows	-9.033585	9.033585	-
53	comfort_convenience_Electrically adjustable seats	5.315421	5.315421	+
51	comfort_convenience_Electrically heated windsh...	4.060606	4.060606	+
9	make_model_Audi A2	0.000000	0.000000	-

3.2.4 Evaluate the Model on Test Data

- Display the values from test data model performance

```
Final Test Evaluation:
R²: 0.8951
MAE: 1323.56
MSE: 3715991.07
```

3.3 Lasso Regression Implementation

3.3.1 Define Alpha Range

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

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

3.3.2.1 Apply Lasso Regularization

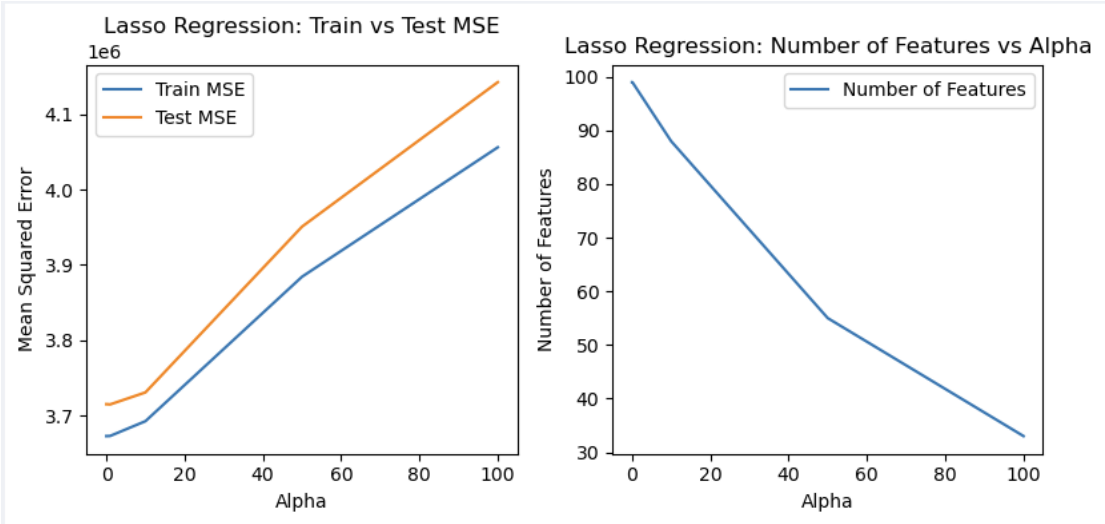
Fits Lasso models for a range of alpha values and returns train and test MSE scores along with the number of features used (non-zero coefficients) for each alpha. We will compare this with the best

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
- Find the train and test scores using mean_squarred_error

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.



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

3.3.3 Fine Tune with Narrow Alpha Range

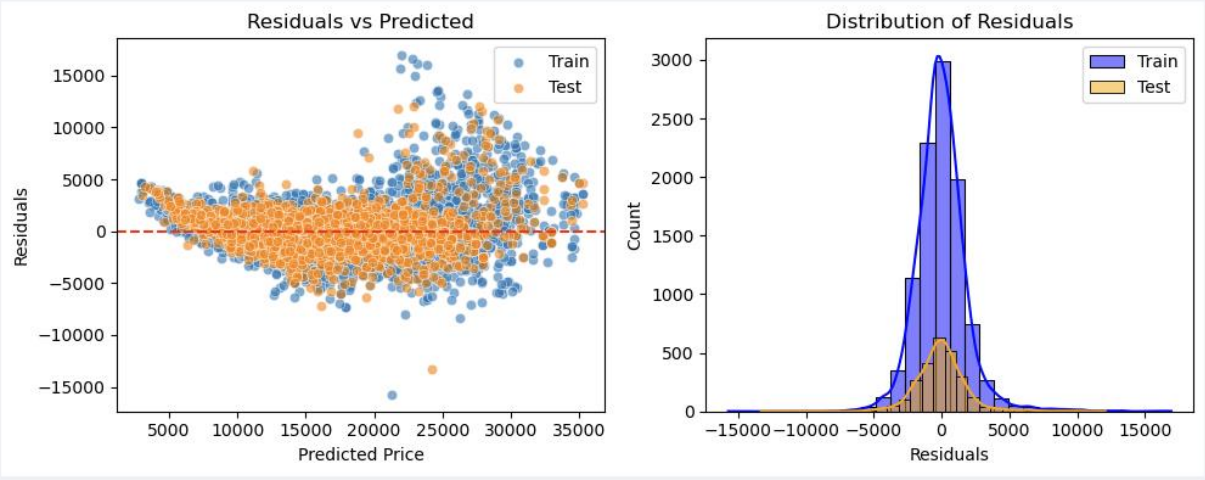
- We use narrow alpha range : np.linspace(0.005, 7, 30) # 30 values between 0.005 and 10
- Fit a LassoCV model to the training data and returns predictions for both train and test sets. Also returns the chosen alpha value
- Initialize LassoCV with narrow alpha range and *cv=KFold(n_splits=5, shuffle=True, random_state=42)*
  - There is no scoring parameter in LassoCV, it uses MSE by default
  - cv=5 is same as default KFold(n_splits=5, shuffle=False)
  - cv=KFold(n_splits=5, shuffle=True, random_state=42)

    - shuffle=True randomizes the order of the data before splitting
    - random_state=42 ensures reproducibility (the same splits every time you run)
- Fit the model on training data
- Predict y, for train and test datasets
- Find best alpha using lassocv.alpha_
- Find best score MAE and MSE for the test predicted and test actual values
- Analyze model performance using the reusable function
- Plot the train and test scores for narrow alpha range using the reusable function

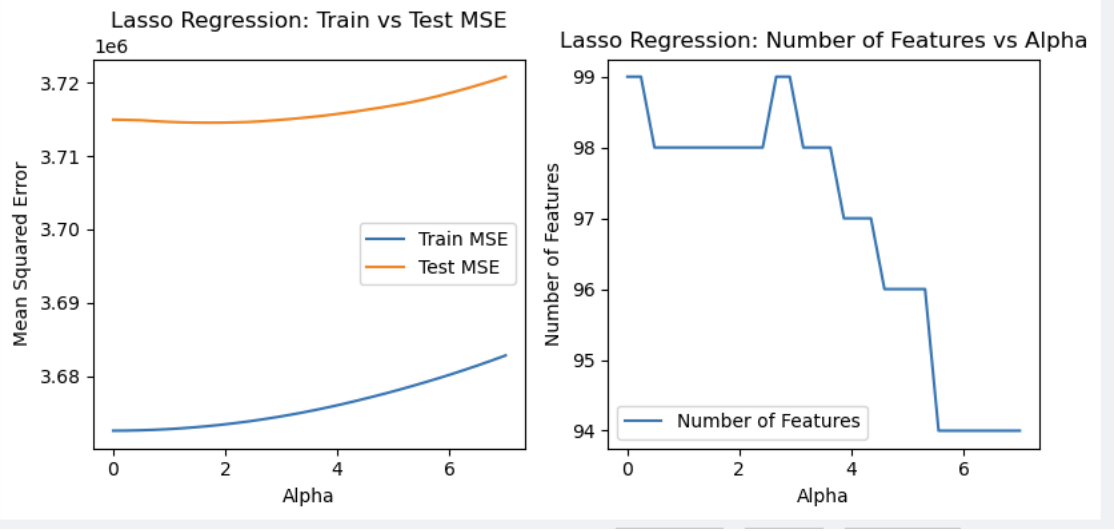
```
Chosen alpha (using LassoCV): 1.934655172413793
Final LassoCV Model - Analyze Model Performance
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1916.608299	5964.193905	3.673387e+06	3.557161e+07	1338.192484	16687.750807	0.088027	0.896723
1	Test	1927.318327	5953.809415	3.714556e+06	3.544785e+07	1325.045784	16695.843505	0.086150	0.895170

```
Final LassoCV Model - Plot of Residuals vs Predicted
Best LassoCV Model Alpha: 1.934655172413793
Best LassoCV Model Score (Negative MSE): -3714555.9329639934 for Alpha 1.934655172413793
Best LassoCV Model Score (Negative MAE): -1325.0457843719294 for Alpha 1.934655172413793
```



```
Finding the best alpha using lasso() tuning and min negative MSE
```




```
We use min Test MSE, to compare with LassoCV later which by default uses MSE
Chosen alpha (Lasso): 1.6934482758620688
Chosen alpha (Lasso) 1.6934482758620688 with minimum Test MSE: 3714545.352458124
```

Inference:

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

3.3.3.1 Run the Model using Best Alpha

- Best Alpha = 1.69
- Use reusable function of lassocv to fit the model on training data and evaluate on test data

- Analyze model performance
- Get the coefficients of the fitted model
- Print the top-n coefficients by absolute value

```
Chosen alpha (using LassoCV): 1.0
Final Lasso Model – Analyze Model Performance
```

	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Training	1916.453306	5964.193905	3.672793e+06	3.557161e+07	1338.680721	16687.750807	0.088078	0.896739
1	Test	1927.344608	5953.809415	3.714657e+06	3.544785e+07	1325.591341	16695.843505	0.086195	0.895167

```
Final Top-N Lasso Model Coefficients by absolute value, with best score alpha:
--------------------------------------------------------------------------------
Top 10 Lasso features and coefficients, with sign
```

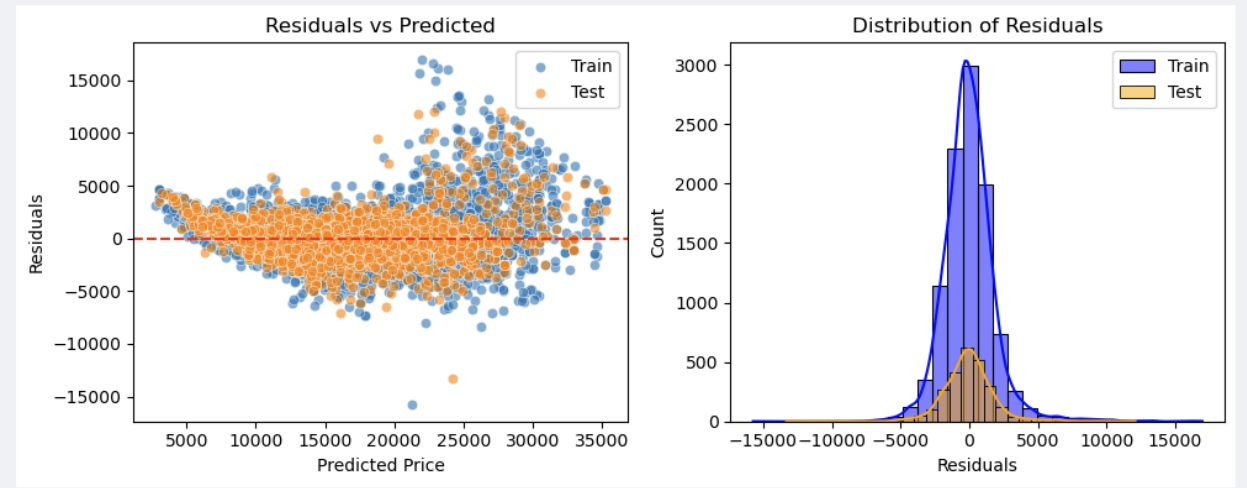
	feature	coef	coef_abs	sign	dropped
12	make_model_Opel Corsa	-1869.271825	1869.271825	-	False
4	hp_kw	1777.459896	1777.459896	+	False
14	make_model_Renault Clio	-1720.817702	1720.817702	-	False
11	make_model_Opel Astra	-1259.253622	1259.253622	-	False
0	km	-1214.449694	1214.449694	-	False
2	age	-1198.522638	1198.522638	-	False
27	gearing_type_Manual	-809.744553	809.744553	-	False
15	make_model_Renault Espace	786.728041	786.728041	+	False
22	type_Used	-708.489207	708.489207	-	False
10	make_model_Audi A3	677.351664	677.351664	+	False

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

	feature	coef	coef_abs	sign	dropped
92	safety_security_Adaptive headlights	-17.616809	17.616809	-	False
57	entertainment_media_Hands-free equipment	14.102644	14.102644	+	False
38	comfort_convenience_Multi-function steering wheel	11.939804	11.939804	+	False
89	safety_security_Alarm system	-9.281042	9.281042	-	False
58	entertainment_media_On-board computer	-7.925456	7.925456	-	False
42	comfort_convenience_Power windows	-6.366409	6.366409	-	False
51	comfort_convenience_Electrically heated windsh...	3.135657	3.135657	+	False
53	comfort_convenience_Electrically adjustable seats	3.006777	3.006777	+	False
8	cons_comb	-0.000000	0.000000	0	True
9	make_model_Audi A2	0.000000	0.000000	0	True

```
--------------------------------------------------------------------------------
Dropped Lasso features and coefficients
--------------------------------------------------------------------------------
```

	feature	coef	coef_abs	sign	dropped
8	cons_comb	-0.0	0.0	0	True
9	make_model_Audi A2	0.0	0.0	0	True



3.3.3.2 Evaluate Model on Test Data

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

```
Final Test Evaluation:
R²: 0.8952
MAE: 1325.05
MSE: 3714555.93
```

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

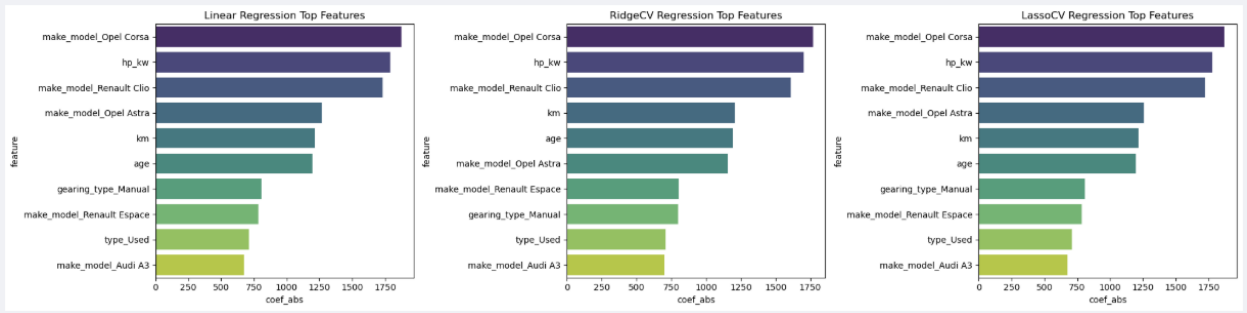
	Model	Alpha	dataset	rmse	std	mse	var	mae	mean	mape	r2
0	Linear	-	Training	1916.395641	5964.193905	3.672572e+06	3.557161e+07	1339.295942	16687.750807	0.088136	0.896745
1	Linear	-	Test	1927.425144	5953.809415	3.714968e+06	3.544785e+07	1326.169542	16695.843505	0.086239	0.895158
2	RidgeCV	110.689655	Training	1918.045885	5964.193905	3.678900e+06	3.557161e+07	1337.148364	16687.750807	0.087747	0.896568
3	RidgeCV	110.689655	Test	1927.496661	5953.809415	3.715243e+06	3.544785e+07	1323.690419	16695.843505	0.085888	0.895150
4	LassoCV	1.0	Training	1916.453306	5964.193905	3.672793e+06	3.557161e+07	1338.680721	16687.750807	0.088078	0.896739
5	LassoCV	1.0	Test	1927.344608	5953.809415	3.714657e+06	3.544785e+07	1325.591341	16695.843505	0.086195	0.895167

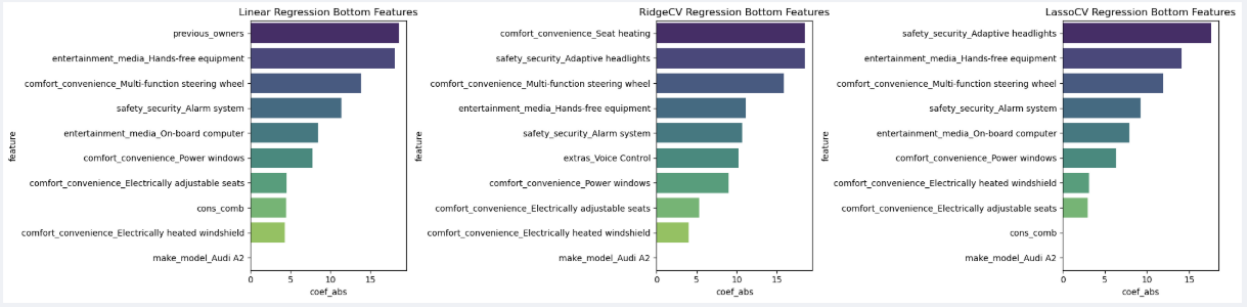
3.4.1.1 Interpretation strategy

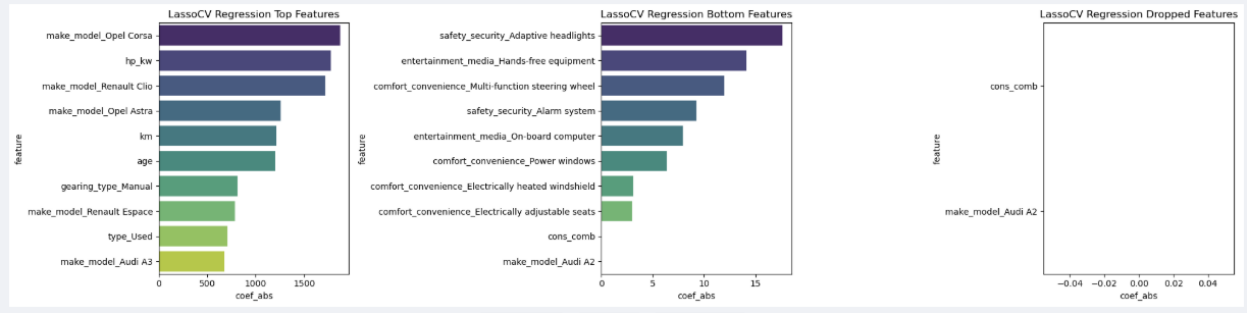
- If Linear Regression has the lowest test error → data is not very multicollinear, and regularization doesn't add much.
- If Ridge improves test performance slightly but reduces variance → multicollinearity was an issue, Ridge is stabilizing coefficients.
- If Lasso performs better (or gives similar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

3.4.2 Compare Coefficients of the 3 models

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







4 Conclusion

4.1 Model Performance Comparison Conclusion

4.1.1 Observations

- All three models perform almost identically
  - **RMSE ≈ 1927**
  - **MAE ≈ 1325**
  - **R^2 ≈ 0.895**

  → Predictive accuracy is effectively the same.
- RidgeCV
  - Chose **α ≈ 110.7**
  - Reduced MAE slightly compared to baseline (**1323 vs 1326**).
  - Regularization didn't change predictive performance much, but it **stabilizes coefficients** (reduces multicollinearity risk).
- LassoCV
  - Chose **α = 1.0**
  - Similar performance to baseline and Ridge.
  - Main benefit: **feature selection** (shrinks/zeroes out less important coefficients).
- Baseline Linear Regression
  - Already very strong (**R^2 ≈ 0.895**).
  - Regularization doesn't improve error metrics, but might affect interpretability.

4.1.2 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).

4.1.3 Predictors and Coefficients Comparison

4.1.3.1 Top Predictors across Models

Feature	Linear Regression	Ridge Regression	Lasso Regression	Interpretation
make_model_Opel Corsa	-1873.56	-1767.95	-1869.27	Strong negative effect; very stable across models.

hp_kw	+1786.34	+1702.84	+1777.46	Strongest positive driver; Ridge shrinks slightly.
make_model_Renault Clio	-1730.77	-1609.58	-1720.82	Consistently strong negative effect.
make_model_Opel Astra	-1268.10	-1155.89	-1259.25	Stable negative contribution.
km	-1215.94	-1204.06	-1214.45	Mileage reduces price; very consistent.
age	-1195.50	-1191.24	-1198.52	Older cars worth less; unchanged across models.
gearing_type_Manual	-807.50	-799.54	-809.74	Manual gearboxes lower price; effect stable.
make_model_Renault Espace	+784.07	+804.01	+786.73	Model-specific premium; Ridge inflates slightly.
type_Used	-710.31	-709.05	-708.49	Used cars discounted; identical across models.
make_model_Audi A3	+676.51	+701.38	+677.35	Positive Audi brand premium; stable.

4.1.4 Bottom Predictors across Models

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

4.1.4.1 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).

4.1.4.2 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).

4.1.4.3 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.

4.1.4.4 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.**

4.1.4.5 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., `cons_comb`, `Audi A2`) that do not meaningfully influence pricing, simplifying analysis and reporting.
- **Actionable Insight for Pricing Strategy**
  - Focus marketing and sales efforts on features that consistently boost value (premium models, high engine power, low mileage).
  - Avoid overcomplicating pricing models with low-impact features; Lasso highlights which features can be safely ignored.
  - Use Ridge when all features matter but multicollinearity exists, ensuring stable coefficient estimates for decision-making.
- **Predictive Accuracy vs Interpretability**
  - All three models (Linear, Ridge, Lasso) predict price with near-identical accuracy ($R^2 \approx 0.895$).
  - **Decision-makers should choose model type based on business need:**
  - Simplified, interpretable pricing → **Lasso**
  - Robust estimates including all features → **Ridge**

- Quick baseline insights → **Linear regression**