Car Price Prediction using Machine Learning Models (Linear Model)

BY: Ashwini KS

EXPECTATIONS:

1) Comment on quality and nature of data
2) What kind of insights did you see. Irregularity.
3) What kind of features influenced the price and which did not.
4) How were models performing? Were they able to do justice to the prediction of prices?
5) What kind of metrics did we use?
6) Business objective & purpose of this assignment? Was the prediction good? Comment on accuracy, and how is okay to have a trade-off? What error range is okay for this?
7) Should we make it more complex so results are better?
8) How to improve this?

Objective:

The business objective of the project is to; Predict the prices of used cars, based on its features and usage in the past.

Data Understanding

Data:

The data is about the prices of used cars. The target is continuous and is numerical. There are a total of 22 columns in the original dataset given, out of which Price column is the target. There are no NULLS in the dataset given. Most of the predictors are categorical, and some columns are stacked as well. These stacked columns need to be unstacked and encoded.

```
# Find the proportion of missing values in each column and handle if found
car_df.info()
car_df.isnull().sum() #data has no missing values

#since the aim is to predict the price, price column is the target => Numerical target
x = car_df.drop('price', axis=1)
y = car_df['price']
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15915 entries, 0 to 15914
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   make_model            15915 non-null  object
 1   body_type             15915 non-null  object
 2   price                 15915 non-null  int64
 3   vat                   15915 non-null  object
 4   km                    15915 non-null  float64
 5   Type                  15915 non-null  object
 6   Fuel                  15915 non-null  object
 7   Gears                 15915 non-null  float64
 8   Comfort_Convenience   15915 non-null  object
 9   Entertainment_Media   15915 non-null  object
 10  Extras                15915 non-null  object
 11  Safety_Security       15915 non-null  object
 12  age                   15915 non-null  float64
 13  Previous_Owners       15915 non-null  float64
 14  hp_kW                 15915 non-null  float64
 15  Inspection_new        15915 non-null  int64
 16  Paint_Type            15915 non-null  object
 17  Upholstery_type       15915 non-null  object
 18  Gearing_Type          15915 non-null  object
 19  Displacement_cc       15915 non-null  float64
 20  Weight_kg             15915 non-null  float64
 21  Drive_chain           15915 non-null  object
 22  cons_comb             15915 non-null  float64
dtypes: float64(8), int64(2), object(13)
memory usage: 2.8+ MB
```

Identify numerical predictors and plot their frequency distributions:

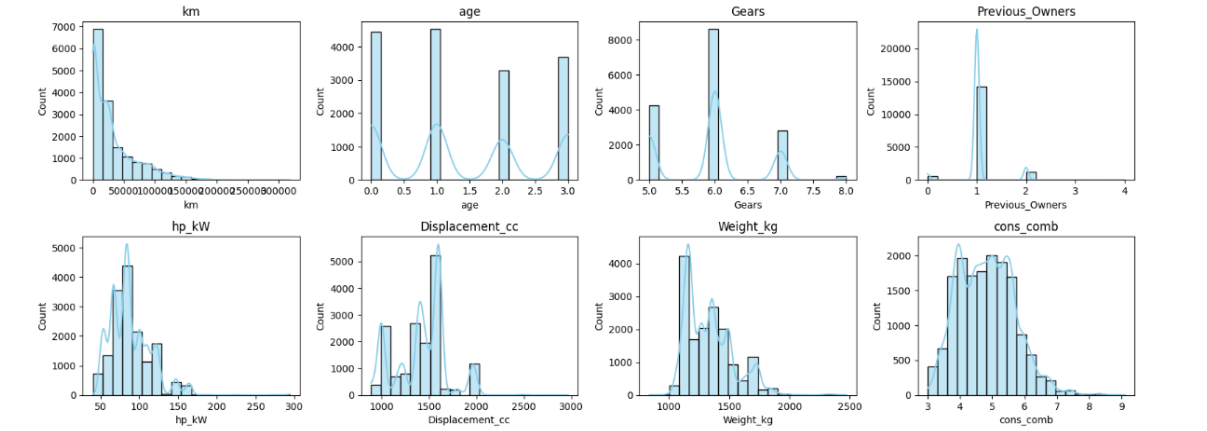
Attached are the numerical columns, with its features. num_cols lists all the numerical columns.

```
# Identify numerical features and plot histograms
x.info()
#numerical cols
num_cols = ['km', 'age', 'Gears','Previous_Owners', 'hp_kW', 'Displacement_cc', 'Weight_kg', 'cons_comb']

#categorical cols
cat_cols = list(set(x.columns) - set(num_cols))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15915 entries, 0 to 15914
Data columns (total 22 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   make_model           15915 non-null  object
 1   body_type            15915 non-null  object
 2   vat                  15915 non-null  object
 3   km                   15915 non-null  float64
 4   Type                 15915 non-null  object
 5   Fuel                 15915 non-null  object
 6   Gears                15915 non-null  float64
 7   Comfort_Convenience  15915 non-null  object
 8   Entertainment_Media  15915 non-null  object
 9   Extras               15915 non-null  object
 10  Safety_Security      15915 non-null  object
 11  age                  15915 non-null  float64
 12  Previous_Owners      15915 non-null  float64
 13  hp_kW                15915 non-null  float64
 14  Inspection_new       15915 non-null  int64
 15  Paint_Type           15915 non-null  object
 16  Upholstery_type      15915 non-null  object
 17  Gearing_Type         15915 non-null  object
 18  Displacement_cc      15915 non-null  float64
 19  Weight_kg            15915 non-null  float64
 20  Drive_chain          15915 non-null  object
 21  cons_comb            15915 non-null  float64
dtypes: float64(8), int64(1), object(13)
memory usage: 2.7+ MB
```

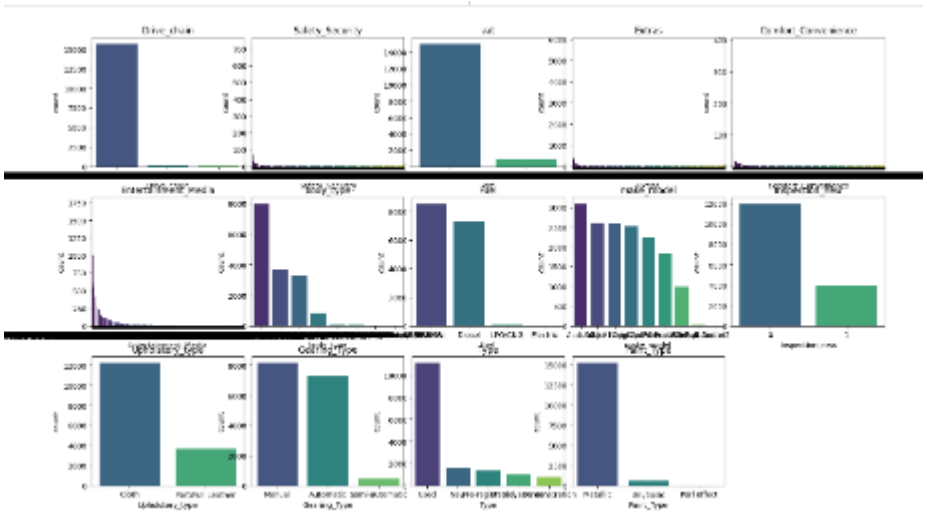
Frequency distribution of the numerical columns.

```
plt.figure(figsize=(16, 12))

for i, col in enumerate(num_cols, 1):
    plt.subplot(4, 4, i)   # adjust layout grid (4x4 here)
    sns.histplot(x[col], bins=20, kde=True, color='skyblue', edgecolor='black')
    plt.title(col)
    plt.tight_layout()

plt.show()
```



Identify categorical predictors and plot their frequency distributions.

The following columns are stacked, even though categorical, they need to be handled in a different way.



analyse_cols = ["Comfort_Convenience", "Entertainment_Media", "Extras", "Safety_Security"]

Fix columns with low frequency values and class imbalances.

```
# Fix columns as needed
#Let us first convert the records in above columns to lists.
for c in analyse_cols:
  x[c] = x[c].fillna('')
  x[c] = x[c].str.split(',')
```

```
x["Comfort_Convenience"] #list of all the columns
```

	Comfort_Convenience
0	[Air conditioning, Armrest, Automatic climate ...
1	[Air conditioning, Automatic climate control, ...
2	[Air conditioning, Cruise control, Electrical ...
3	[Air suspension, Armrest, Auxiliary heating, E...
4	[Air conditioning, Armrest, Automatic climate ...
...	...
15910	[Air conditioning, Automatic climate control, ...
15911	[Air conditioning, Automatic climate control, ...
15912	[Air conditioning, Armrest, Automatic climate ...
15913	[Air conditioning, Automatic climate control, ...
15914	[Air conditioning, Automatic climate control, ...

15915 rows × 1 columns

dtype: object

In order to fix the above type of columns, let us first convert the stacks into lists, and then multibinarize it. MultiLabelBinarizer() : This one helps to multi encode.

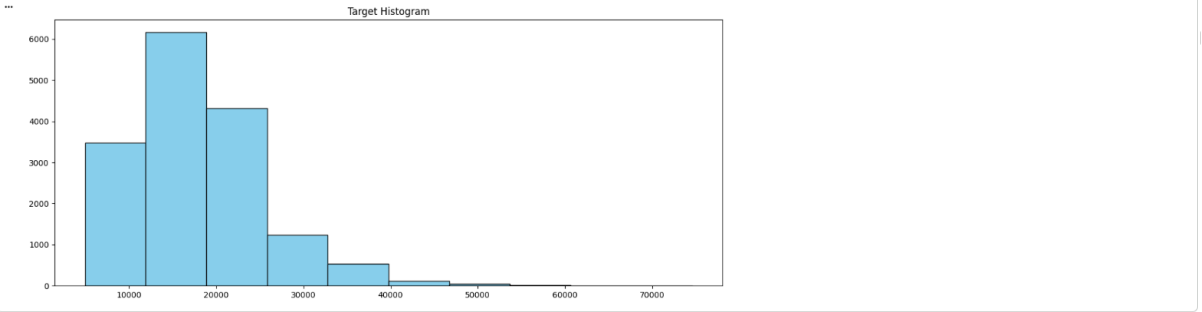
```
mlb = MultiLabelBinarizer()
for c in analyse_cols:
  features_encoded = mlb.fit_transform(x[c])
  features_encoded = pd.DataFrame(features_encoded, columns=mlb.classes_)
  features_encoded = features_encoded.loc[:, features_encoded.sum() > 100]
  x = pd.concat([x, features_encoded], axis=1)
  x = x.drop(c, axis=1)
x
```

	make_model	body_type	vat	km	Type	Fuel	Gears	age	Previous_Owners	hp_kW	...	LED Headlights	Lane departure warning system	Passenger-side airbag	Power steering	Rear airbag	Side airbag	Tire pressure monitoring system	Traction control	Traffic sign recognition	Xenon headlights
0	Audi A1	Sedans	VAT deductible	56013.000000	Used	Diesel	7.0	3.0	2.0	66.0	...	0	0	1	1	0	1	1	1	0	1
1	Audi A1	Sedans	Price negotiable	80000.000000	Used	Benzine	7.0	2.0	1.0	141.0	...	0	0	1	1	0	1	1	1	0	1
2	Audi A1	Sedans	VAT deductible	83450.000000	Used	Diesel	7.0	3.0	1.0	85.0	...	0	0	1	1	0	1	1	1	0	0
3	Audi A1	Sedans	VAT deductible	73000.000000	Used	Diesel	6.0	3.0	1.0	66.0	...	0	0	1	1	0	1	1	0	0	0
4	Audi A1	Sedans	VAT deductible	16200.000000	Used	Diesel	7.0	3.0	1.0	66.0	...	0	0	1	1	0	1	1	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15910	Renault Espace	Van	VAT deductible	1647.362609	New	Diesel	6.0	0.0	1.0	147.0	...	1	1	1	1	1	1	1	1	1	0
15911	Renault Espace	Van	VAT deductible	9900.000000	Used	Benzine	7.0	0.0	1.0	165.0	...	1	1	1	1	0	1	1	1	1	0
15912	Renault Espace	Van	VAT deductible	15.000000	Pre-registered	Diesel	6.0	0.0	1.0	146.0	...	1	1	1	1	0	1	0	1	1	0
15913	Renault Espace	Van	VAT deductible	10.000000	Pre-registered	Diesel	6.0	0.0	1.0	147.0	...	1	0	1	1	0	1	1	0	1	0
15914	Renault Espace	Van	VAT deductible	1647.362609	Demonstration	Benzine	6.0	0.0	1.0	165.0	...	0	0	1	1	0	1	1	0	1	0

15915 rows × 101 columns

Identify target variable and plot the frequency distributions. Apply necessary transformations.

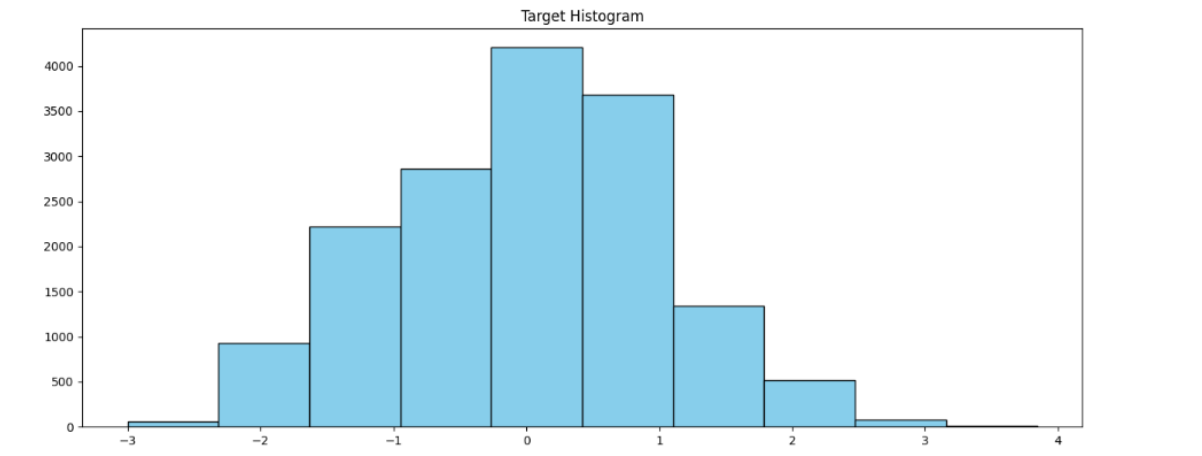
```
# Plot histograms for target feature
#target feature is the price
plt.figure(figsize = (15, 6))
plt.hist(y, bins=10, range=None, color='skyblue', edgecolor='black')
plt.title("Target Histogram")
plt.show()
#plot shows that target seem to be right skewed
```



The target variable seems to be skewed. Perform suitable transformation on the target.

To fix the right skew, let us use transformation technique.

```
# Transform the target feature
y.skew() # we can see that the skew number is 0.999, and since this is positively skewed, transforms like box-cox or Yeo-Johnson will help.
#sklearn preprocessing already has Yeo-Johnson transform technique. Using that
pt = PowerTransformer(method='yeo-johnson')
y = pt.fit_transform(y.values.reshape(-1, 1))

plt.figure(figsize = (15, 6))
plt.hist(y, bins=10, range=None, color='skyblue', edgecolor='black')
plt.title("Target Histogram")
plt.show()
```

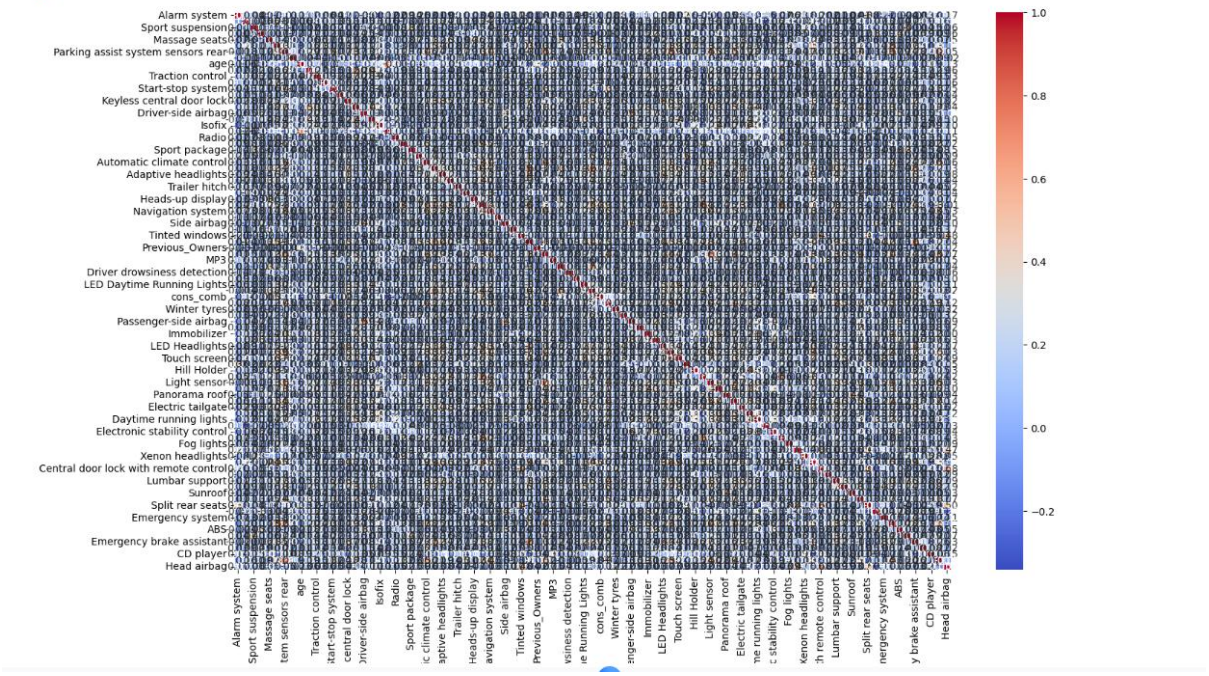


Plot the correlation map between features and target variable.

From the attached image, since the correlation heatmap is too cluttered, let us get the triangular map.

```
corr_matrix = x_num.corr()
plt.figure(figsize=(16, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
```

```
<Axes: >
```




```
#there are 91 columns and all there would may or may not be necessary
#also 2 or more columns may carry redundant information
upper = abs(corr_matrix).where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]

x_num = x_num.drop(columns=to_drop)
```

```
x_num
```

	Alarm system	Roof rack	Sport suspension	Power steering	Massage seats	Heated steering wheel	Parking assist system sensors rear	Cab or rented Car	age	Bluetooth	...	Split rear seats	Central door lock	Emergency system	Seat heating	ABS	Sport seats	Emergency brake assistant	Air conditioning	CD player	Head airbag
0	0	0	0	1	0	0	1	0	3.0	1	...	0	1	0	1	1	0	0	1	0	0
1	0	0	1	1	0	0	1	0	2.0	1	...	0	1	0	0	1	1	0	1	0	1
2	0	0	0	1	0	0	1	0	3.0	0	...	0	1	0	1	1	0	0	1	0	0
3	1	0	0	1	0	0	0	0	3.0	1	...	1	0	1	1	1	1	0	0	1	1
4	0	0	1	1	0	0	1	0	3.0	1	...	0	1	0	0	1	0	0	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15910	0	0	0	1	1	0	1	0	0.0	1	...	1	1	0	0	1	0	1	1	0	1
15911	0	0	0	1	0	1	1	0	0.0	1	...	0	1	0	1	1	0	1	1	0	0
15912	0	0	0	1	1	1	1	0	0.0	1	...	0	1	0	1	1	0	1	1	0	0
15913	0	0	0	1	0	0	0	0	0.0	1	...	0	0	0	0	1	0	1	1	0	0
15914	0	0	0	1	0	0	0	0	0.0	0	...	0	0	0	1	1	0	1	1	0	0

15915 rows × 90 columns

Analyse correlation between categorical features and target variable.

```
# Comparing average values of target for different categories
cat_cols
#lets group and compute average
#group = car_df.groupby("make_model")["price"].mean()
#group
```

```
['Comfort_Convenience',
 'Extras',
 'Gearing_Type',
 'Drive_chain',
 'body_type',
 'Inspection_new',
 'Type',
 'vat',
 'Safety_Security',
 'Entertainment_Media',
 'Paint_Type',
 'Fuel',
 'make_model',
 'Upholstery_type']
```

```
car_df["Upholstery_type"].unique()
```

```
array(['Cloth', 'Part/Full Leather'], dtype=object)
```

```
x_encoded = pd.get_dummies(car_df[cat_cols], drop_first=True)
```

Identify potential outliers in the data.

```
# Outliers present in each column
#Let us define IQR method to get the outliers
num_cols = x_num.columns
num_cols
```

```
Index(['Alarm system', 'Roof rack', 'Sport suspension', 'Power steering',
       'Massage seats', 'Heated steering wheel',
       'Parking assist system sensors rear', 'Cab or rented Car', 'age',
       'Bluetooth', 'Traction control', 'Electrically heated windshield',
       'Start-stop system', 'Auxiliary heating', 'Keyless central door lock',
       'Hands-free equipment', 'Driver-side airbag', 'USB', 'Isofix', 'km',
       'Radio', 'Gears', 'Sport package', 'Adaptive Cruise Control',
       'Automatic climate control', 'Traffic sign recognition',
       'Adaptive headlights', 'Electrically adjustable seats', 'Trailer hitch',
       'Cruise control', 'Heads-up display', 'hp_kW', 'Navigation system',
       'Shift paddles', 'Side airbag', 'Sound system', 'Tinted windows',
       'Lane departure warning system', 'Previous_Owners', 'Rain sensor',
       'MP3', 'Parking assist system self-steering',
       'Driver drowsiness detection', 'Ski bag', 'LED Daytime Running Lights',
       'Blind spot monitor', 'cons_comb', 'Armrest', 'Winter tyres',
       'Digital radio', 'Passenger-side airbag', 'On-board computer',
       'Immobilizer', 'Power windows', 'LED Headlights',
       'Parking assist system sensors front', 'Touch screen',
       'Seat ventilation', 'Hill Holder', 'Weight_kg', 'Light sensor',
       'Electrical side mirrors', 'Panorama roof', 'Leather steering wheel',
       'Electric tailgate', 'Tire pressure monitoring system',
       'Daytime running lights', 'Alloy wheels',
       'Electronic stability control', 'Displacement_cc', 'Fog lights',
       'Rear airbag', 'Xenon headlights', 'Parking assist system camera',
       'Central door lock with remote control', 'Catalytic Converter',
       'Lumbar support', 'Voice Control', 'Sunroof',
       'Multi-function steering wheel', 'Split rear seats',
       'Central door lock', 'Emergency system', 'Seat heating', 'ABS',
       'Sport seats', 'Emergency brake assistant', 'Air conditioning',
       'CD player', 'Head airbag'],
      dtype='object')
```

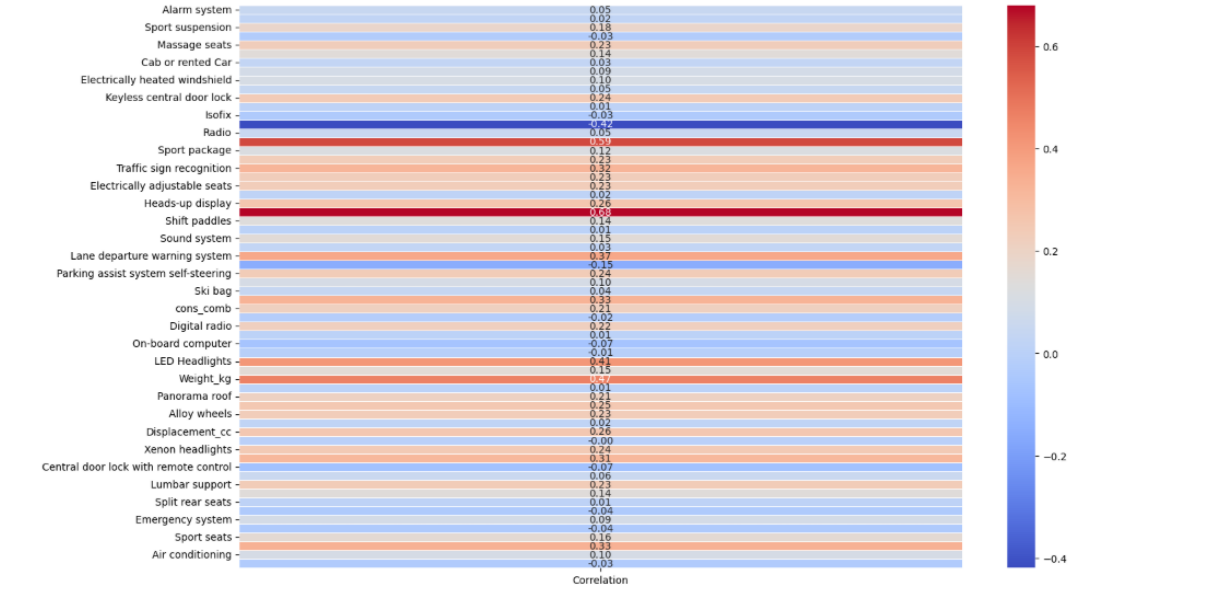
```
#using IQR method, try and find the outliers
outlier_nums = {}
for col in num_cols:
  Q1 = x_num[col].quantile(0.25)
  Q3 = x_num[col].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  outliers = x_num[(x_num[col] < lower_bound) | (x_num[col] > upper_bound)]
  if len(outliers) > 0:
    outlier_nums[col] = len(outliers)
outlier_nums
```

```
 'Cab or rented Car': 310,
 'Bluetooth': 3230,
 'Electrically heated windshield': 944,
 'Auxiliary heating': 266,
 'Keyless central door lock': 2794,
 'Driver-side airbag': 1114,
 'Isofix': 3226,
```

```
import numpy as np
outliers = list(outlier_nums.keys())
x_outliers = pd.DataFrame()
for cols in outliers:
  x_outliers[cols] = x_num[cols]
corr_y = x_outliers.corrwith(pd.Series(y.squeeze(), index=x_outliers.index))
corr_y = pd.DataFrame(corr_y, columns=['Correlation'])
```

```
plt.figure(figsize=(16, 10))
sns.heatmap(corr_y, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
```

```
<Axes: >
```



Let us conduct winsorization on the above columns depending on their correlation with the target. We may remove such rows which are not suitable as well.

```
corr_y[abs(corr_y['Correlation']) < 0.30] #these are the columns which have outliers and not really that well correlated with price. Thus we can get rid of/winsorise outliers here.
wins_cols = list(corr_y.index) #get all the columns that can undergo winsorisation
```

Handle the outliers suitably.

```
# Handle outliers

#now understand handle the outliers
for col in wins_cols:
  Q1 = x_num[col].quantile(0.25)
  Q3 = x_num[col].quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  x_num[col] = x_num[col].clip(lower=lower_bound, upper=upper_bound)
```

Feature Engineering

Analysis and feature engineering on ['Comfort_Convenience', 'Entertainment_Media', 'Extras', 'Safety_Security'].

These columns contains lists of features present. Decide on how to include these features in the predictors.

```python
# Check unique values in each feature spec columns
unique_dictionary = {}
for col in x_num_cols:
  unique_dictionary[col] = x_num[col].unique()
unique_dictionary
```

```
{'Alarm system': array([0]),
 'Roof rack': array([0]),
 'Sport suspension': array([0]),
 'Power steering': array([1]),
 'Massage seats': array([0]),
 'Heated steering wheel': array([0]),
 'Parking assist system sensors rear': array([1, 0]),
 'Cab or rented Car': array([0]),
 'age': array([3., 2., 1., 0.]),
 'Bluetooth': array([1]),
 'Traction control': array([1, 0]),
 'Electrically heated windshield': array([0]),
 'Start-stop system': array([1, 0]),
 'Auxiliary heating': array([0]),
 'Keyless central door lock': array([0]),
 'Hands-free equipment': array([1, 0]),
 'Driver-side airbag': array([1]),
 'USB': array([0, 1]),
 'Isofix': array([1]),
 'km': array([5.6013e+04, 8.0000e+04, 8.3450e+04, ..., 2.8640e+03, 1.5060e+03,
        5.7000e+01]),
 'Radio': array([1]),
 'Gears': array([7. , 6. , 5. , 7.5]),
 'Sport package': array([0]),
 'Adaptive Cruise Control': array([0]),
 'Automatic climate control': array([1, 0]),
 'Traffic sign recognition': array([0]),
 'Adaptive headlights': array([0]),
 'Electrically adjustable seats': array([0]),
 'Trailer hitch': array([0]),
 'Cruise control': array([1, 0]),
 'Heads-up display': array([0]),
 'hp_kW': array([ 66. , 141. ,  85. ,  70. ,  92. , 112. ,  60. ,  71. ,  67. ,
        110. ,  93. , 147. ,  86. , 140. ,  87. ,  81. ,  82. , 135. ,
        132. , 100. ,  96. , 158.5, 150. , 137. , 133. ,  77. , 101. ,
```

#got all the unique values in each col

Out of these features, we will check the ones which are present in most of the cars or are absent from most of the cars. These kinds of features can be removed as they just increase the dimensionality

without explaining the variance.

```
# Drop features from df
for col in zero_val:
  x_num = x_num.drop(col, axis=1)
```

```
value = np.array([0, 1])
one_val = [k for k, v in unique_dictionary.items() if np.array_equal(v, value)]
#get the sum of true and check how many sums are present => In total there are 15,915 number of records
for o in one_val:
  print(o, x_num[o].sum())

value_2 = np.array([0, 1])
val_one = [k for k, v in unique_dictionary.items() if np.array_equal(v, value_2)]
#get the sum of true and check how many sums are present => In total there are 15,915 number of records
for o in val_one:
  print(o, x_num[o].sum())

#all the column seems to have enough true values to be considered
```

```
USB 9019
MP3 6258
LED Daytime Running Lights 5910
Parking assist system sensors front 6103
Touch screen 4050
CD player 5024
USB 9019
MP3 6258
LED Daytime Running Lights 5910
Parking assist system sensors front 6103
Touch screen 4050
CD player 5024
```

Perform feature encoding.

```
# Encode features
encoder = OneHotEncoder(sparse_output=False, drop='first')
cat_encoded = pd.DataFrame(index=x.index)

for col in cat_cols:
    if col not in x.columns:
        print(f"Skipping missing column: {col}")
        continue

    cats = encoder.fit_transform(x[[col]])
    cats = pd.DataFrame(
        cats,
        columns=encoder.get_feature_names_out([col]),
        index=x.index
    )

    cat_encoded = pd.concat([cat_encoded, cats], axis=1)

# Merge numeric + encoded features
x_num_merged = pd.concat([x_num, cat_encoded], axis=1)

# Align target with same index
x_num_merged = pd.concat([x_num_merged, y], axis=1)
```

```
Skipping missing column: Comfort_Convenience
Skipping missing column: Extras
Skipping missing column: Safety_Security
Skipping missing column: Entertainment_Media
```

Then we merge the data altogether so we can use it accordingly

Split the data into training and testing sets.

```
# Split data
y = x_df['price']
x_df = x_df.drop('price', axis=1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_df, y)
```

Scale the features.

```
# Scale features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Linear Regression Model

Build and fit a basic linear regression model. Perform evaluation using suitable metrics.

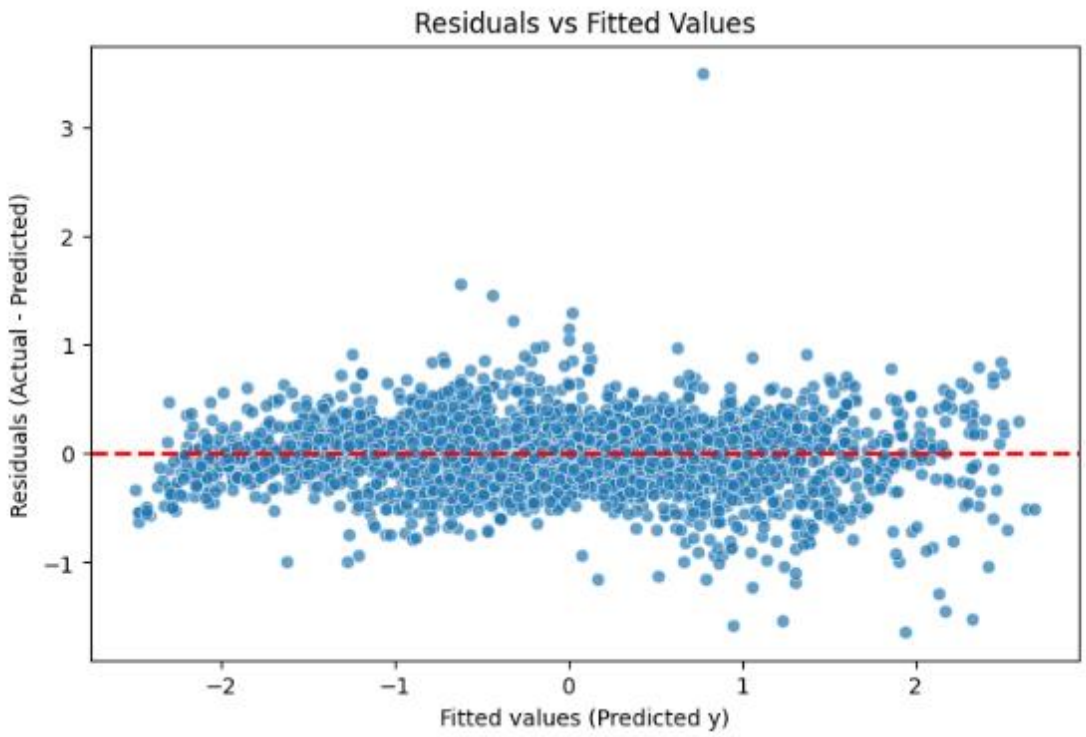
```
# Initialise and train model
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train_scaled, y_train)
y_train_pred = lr.predict(x_train_scaled)
y_test_pred = lr.predict(x_test_scaled)
```

```
# Evaluate the model's performance

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
#mean squared errors
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
print("Train MSE:", train_mse)
print("Test MSE:", test_mse)

#mean absolute error
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
print("Train MAE:", train_mae)
print("Test MAE:", test_mae)

#r-squared
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
print("Train r2:", train_r2)
print("Test r2:", test_r2)
```

```
Train MSE: 0.07778227959188332
Test MSE: 0.08284999580644377
Train MAE: 0.20756701615295906
Test MAE: 0.2091924280846787
Train r2: 0.9210713629960344
Test r2: 0.9206089294872207
```

Analyse residuals and check other assumptions of linear regression.

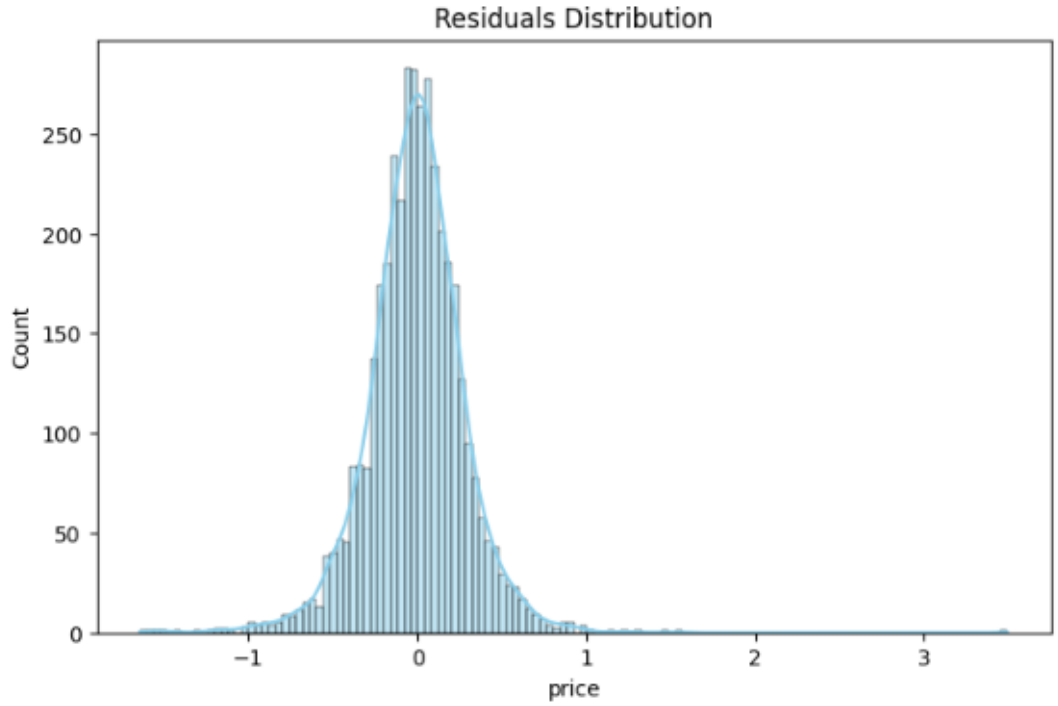
```
# Linearity check: Plot residuals vs fitted values
residuals = y_test_pred - y_test

# Plot residuals vs fitted values
plt.figure(figsize=(8,5))
sns.scatterplot(x=y_test_pred, y=residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--', linewidth=2)
plt.xlabel('Fitted values (Predicted y)')
plt.ylabel('Residuals (Actual - Predicted)')
plt.title('Residuals vs Fitted Values')
plt.show()
#the residues are evenly and non-pattern distributed across 0. This has good linearity
```



Above plot shows that the residuals are linear.

Check normality in residual distribution

```
# Check the normality of residuals by plotting their distribution
plt.figure(figsize=(8,5))
sns.histplot(residuals, kde=True, color='skyblue', edgecolor='black')
plt.title("Residuals Distribution")
plt.show()
#it is almost normally distributed
```



Check multicollinearity using Variance Inflation Factor (VIF) and handle features with high VIF.

```
# Check for multicollinearity and handle
#VIF tells how much multicollinearity exists
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["Feature"] = x_df.columns
vif_data["VIF"] = [variance_inflation_factor(x_df.values, i) for i in range(x_df.shape[1])]

print(vif_data)
```

```
                            Feature       VIF
0                     Power steering  0.000000
1   Parking assist system sensors rear  2.619312
2                                age  4.135225
3                          Bluetooth  0.000000
4                  Traction control  1.335939
..                              ...       ...
74           make_model_Opel Insignia  5.800883
75            make_model_Renault Clio  3.093574
76          make_model_Renault Duster  2.961834
77          make_model_Renault Espace  9.398642
78  Upholstery_type_Part/Full Leather  1.473714

[79 rows x 2 columns]
```

```
#features with vif > 10 are considered to have high multicolinearity and vif 1 to 5 are considered to have good linearity
vif_data[vif_data['VIF'] > 5]
#no vifs greater than 10. The above can still be solved using regularisation
```

	Feature	VIF
15	hp_kW	7.869446
39	Displacement_cc	6.559201
58	body_type_Van	6.780600
63	Type_Used	5.660040
67	Fuel_Diesel	6.727962
74	make_model_Opel Insignia	5.800883
77	make_model_Renault Espace	9.398642

Ridge Regression Implementation

3.2.1 [2 marks]

Define a list of random alpha values

```
# List of alphas to tune for Ridge regularisation
# Lets define alpha in a standard manner
alphas = np.logspace(-3, 3, 20)
```

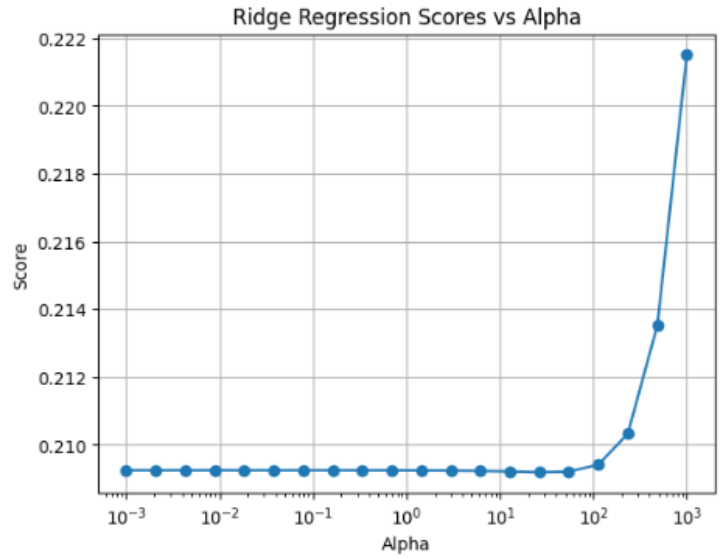
3.2.2 [4 marks]

Apply Ridge Regularisation and find the best value of alpha from the list

```
# Applying Ridge regression
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
ridge = Ridge()
param = {'alpha': alphas}
tune = GridSearchCV(ridge, param, scoring='neg_mean_absolute_error', cv=5) #5-fold, and scoring is neg mae
tune.fit(x_train_scaled, y_train)
```

```
GridSearchCV
best_estimator_:
Ridge
Ridge
```

```
# Plot train and test scores against alpha
results = tune.cv_results_
alpha_values = results["param_alpha"] #gives the alphas used
scores_test = -results['mean_test_score'] #gives the test scores

# Plot line plot
plt.plot(alphas, scores_test, marker='o')
plt.xscale('log')     # optional if alphas are in log scale
plt.xlabel('Alpha')
plt.ylabel('Score')
plt.title('Ridge Regression Scores vs Alpha')
plt.grid(True)
plt.show()
```




```
# Best alpha value
best_alpha = tune.best_params_['alpha']
print("best alpha : ",best_alpha)

# Best score (negative MAE)
best_score = tune.best_score_
print("best mae score : ",-best_score)
```

```
best alpha :  26.366508987303554
best mae score :  0.20918959610482676
```

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.

The following seems to be the best alpha and the score:

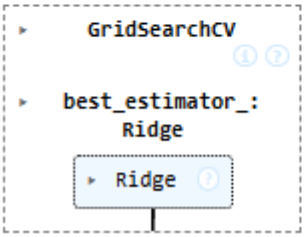
best alpha : 26.366508987303554

best mae score : 0.20918959610482676

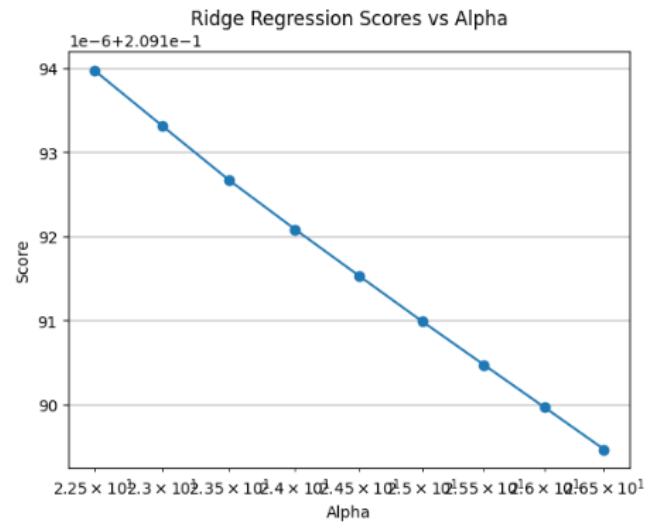
Fine tune by taking a closer range of alpha based on the previous result.

```
# Take a smaller range of alpha to test
#best alpha is 26.3 => lets space it accordingly
alphas = [22.5, 23, 23.5, 24, 24.5, 25, 25.5, 26, 26.5]
```

```
# Applying Ridge regression
ridge = Ridge()
param = {'alpha': alphas}
tune = GridSearchCV(ridge, param, scoring='neg_mean_absolute_error', cv=5)
tune.fit(x_train_scaled, y_train)
```




```
# Plot train and test scores against alpha
results = tune.cv_results_
alpha_values = results["param_alpha"] #gives the alphas used
scores_test = -results['mean_test_score'] #gives the test scores

# Plot line plot
plt.plot(alphas, scores_test, marker='o')
plt.xscale('log')     # optional if alphas are in log scale
plt.xlabel('Alpha')
plt.ylabel('Score')
plt.title('Ridge Regression Scores vs Alpha')
plt.grid(True)
plt.show()

# Best alpha value
print("Best Alpha:", tune.best_params_['alpha'])


# Best score (negative MAE)
print("Best Score:", -tune.best_score_)
```




```
Best Alpha: 26.5
Best Score: 0.20918946558965454
```

Best alpha and score from the curve: Best Alpha: 26.5

Best Score: 0.20918946558965454

```python
# Set best alpha for Ridge regression
# Fit the Ridge model to get the coefficients of the fitted model
ridge_model = Ridge(tune.best_params_['alpha'])
ridge_model.fit(x_train_scaled, y_train)
coefficients = ridge_model.coef_
intercept = ridge_model.intercept_
```

```python
# Show the coefficients for each feature
x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_df.columns)
coef_df_ridge = pd.DataFrame({
    'Feature': x_train_scaled.columns,
    'Coefficient': ridge_model.coef_
}).sort_values(by='Coefficient', ascending=False)

coef_df_ridge
```

	Feature	Coefficient
15	hp_kW	0.287401
77	make_model_Renault Espace	0.103801
71	make_model_Audi A3	0.083043
49	Gearing_Type_Semi-automatic	0.052481
67	Fuel_Diesel	0.042279
...	...	...
72	make_model_Opel Astra	-0.211484
10	km	-0.226207
2	age	-0.242212
75	make_model_Renault Clio	-0.313404
73	make_model_Opel Corsa	-0.364583

79 rows × 2 columns

```
# Evaluate the Ridge model on the test data
y_pred_test = ridge_model.predict(x_test_scaled)
#predicted the results
y_pred_train = ridge_model.predict(x_train_scaled)
```

```
mae_train_ridge = mean_absolute_error(y_train, y_pred_train)
mse_train_ridge = mean_squared_error(y_train, y_pred_train)
r2_train_ridge = r2_score(y_train, y_pred_train)

print(f"Mean Absolute Error (MAE): {mae_train_ridge:.4f}")
print(f"Mean Squared Error (MSE): {mse_train_ridge:.4f}")
print(f"R² Score: {r2_train_ridge:.4f}")
```

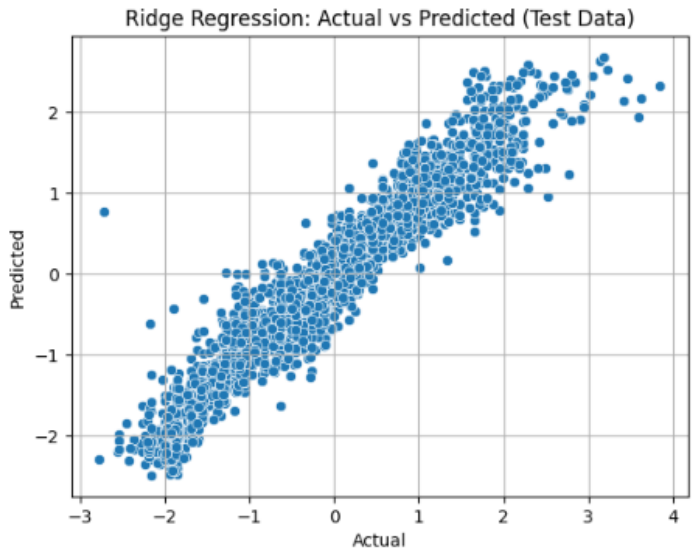
```
Mean Absolute Error (MAE): 0.2075
Mean Squared Error (MSE): 0.0778
R² Score: 0.9211
```

```
#now lets find the scores
mae_ridge = mean_absolute_error(y_test, y_pred_test)
mse_ridge = mean_squared_error(y_test, y_pred_test)
r2_ridge = r2_score(y_test, y_pred_test)

print(f"Mean Absolute Error (MAE): {mae_ridge:.4f}")
print(f"Mean Squared Error (MSE): {mse_ridge:.4f}")
print(f"R² Score: {r2_ridge:.4f}")
#two scores are comparable
```

```
Mean Absolute Error (MAE): 0.2092
Mean Squared Error (MSE): 0.0829
R² Score: 0.9206
```

Now let us plot Actual vs Predicted values on test data.

```
sns.scatterplot(x=y_test, y=y_test_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Ridge Regression: Actual vs Predicted (Test Data)")
plt.grid(True)
plt.show()
#they look good!
```



Lasso Regression Implementation

Define a list of random alpha values

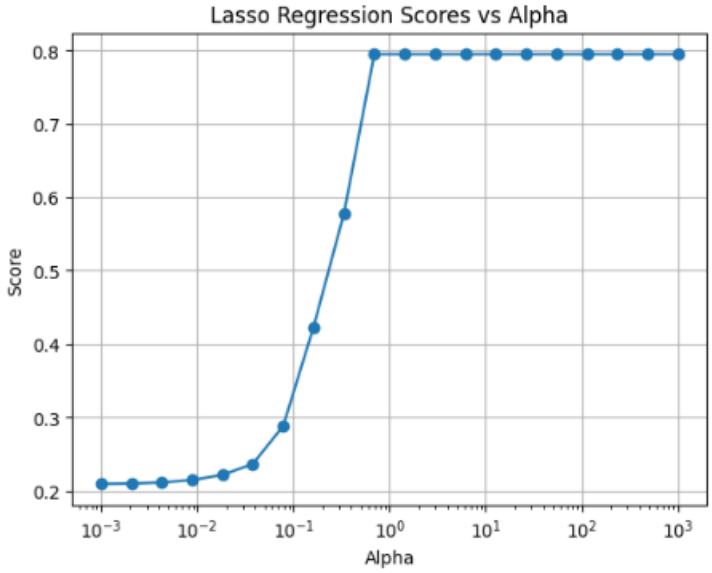
```
# List of alphas to tune for Lasso regularisation
alphas = np.logspace(-3, 3, 20)
```

Apply Ridge Regularisation and find the best value of alpha from the list

```
# Initialise Lasso regression model
from sklearn.linear_model import Lasso
lasso = Lasso()
param = {'alpha': alphas}
tune = GridSearchCV(lasso, param, scoring='neg_mean_absolute_error', cv=5)
tune.fit(x_train_scaled, y_train)
```

```
GridSearchCV
best_estimator_:
Lasso
Lasso
```

```
# Plot train and test scores against alpha
results = tune.cv_results_
alpha_values = results["param_alpha"] #gives the alphas used
scores_test = -results['mean_test_score'] #gives the test scores

# Plot line plot
plt.plot(alphas, scores_test, marker='o')
plt.xscale('log')     # optional if alphas are in log scale
plt.xlabel('Alpha')
plt.ylabel('Score')
plt.title('Lasso Regression Scores vs Alpha')
plt.grid(True)
plt.show()
```



best alpha : 0.001

best mae score : 0.20925853699561273

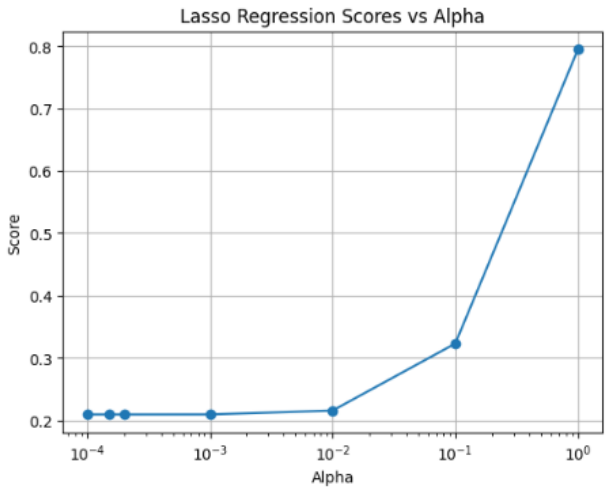
```
# Best alpha value
best_alpha = tune.best_params_['alpha']
print("best alpha : ",best_alpha)

# Best score (negative MAE)
best_score = tune.best_score_
print("best mae score : ",-best_score)
```

Fine tune by taking a closer range of alpha based on the previous result.

```
# List of alphas to tune for Lasso regularization
alphas = [0.0001,0.00015, 0.00020,.001, 0.01, 0.1, 1]
```

```
# Tuning Lasso hyperparameters
lasso = Lasso()
param = {'alpha': alphas}
tune = GridSearchCV(lasso, param, scoring='neg_mean_absolute_error', cv=5)
tune.fit(x_train_scaled, y_train)
```


```
# Plot train and test scores against alpha
results = tune.cv_results_
alpha_values = results["param_alpha"] #gives the alphas used
scores_test = -results['mean_test_score'] #gives the test scores

# Plot line plot
plt.plot(alphas, scores_test, marker='o')
plt.xscale('log')     # optional if alphas are in log scale
plt.xlabel('Alpha')
plt.ylabel('Score')
plt.title('Lasso Regression Scores vs Alpha')
plt.grid(True)
plt.show()
```



```python
# Best alpha value
best_alpha = tune.best_params_['alpha']
print("best alpha : ",best_alpha)

# Best score (negative MAE)
best_score = tune.best_score_
print("best mae score : ",-best_score)
```

```
best alpha :  0.0002
best mae score :  0.20919105442981362
```

```python
# Set best alpha for Lasso regression
lasso_model = Lasso(best_alpha)

# Fit the Lasso model on scaled training data
lasso_model.fit(x_train_scaled, y_train)

# Get the coefficients of the fitted model
x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_df.columns)
coef_df_lasso = pd.DataFrame({
    'Feature': x_train_scaled.columns,
    'Coefficient': ridge_model.coef_
}).sort_values(by='Coefficient', ascending=False)
```

```python
# Check the coefficients for each feature
coef_df_lasso
```

	Feature	Coefficient
24	hp_kW	0.285592
66	make_model_Renault Espace	0.098124
60	make_model_Audi A3	0.082628
56	Gearing_Type_Semi-automatic	0.058579
48	Fuel_Diesel	0.045225
...	...	...
61	make_model_Opel Astra	-0.215405
39	km	-0.219700
13	age	-0.245139
64	make_model_Renault Clio	-0.318072
62	make_model_Opel Corsa	-0.364837

79 rows × 2 columns

Regularisation Comparison & Analysis

Compare the evaluation metrics for each model.

```
# Compare metrics for each model
#LR: 0.9206089294872207
#RidgeReg: 0.9206
#LassoRidge: 0.9206
LR = [test_mse, test_mae, test_r2]
ridgeR = [mse_ridge, mae_ridge, r2_ridge]
lassoR = [mse_lasso, mae_lasso, r2_lasso]
index = ['MSE', 'MAE', 'r2_score']
col = ['Linear Regression', 'Ridge', 'Lasso']

evaluation_metrics = pd.DataFrame( [LR, ridgeR, lassoR],   # data rows
    columns=index,          # metrics as columns
    index=col               # model names as index
).T
evaluation_metrics
```

	Linear Regression	Ridge	Lasso
MSE	0.082850	0.082880	0.082838
MAE	0.209192	0.209223	0.209151
r2_score	0.920609	0.920580	0.920620

Compare the coefficients for the three models.

Also visualise a few of the largest coefficients and the coefficients of features dropped by Lasso.

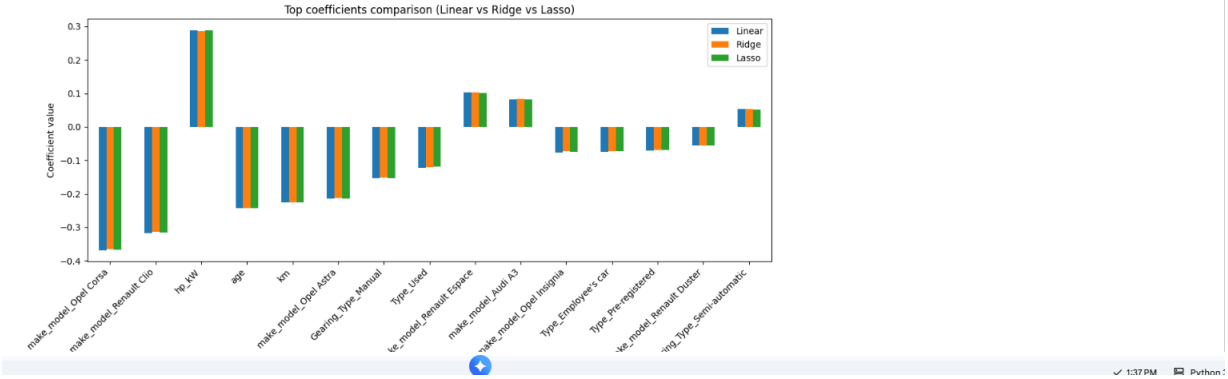
```
# Compare highest coefficients and coefficients of eliminated features
#lasso: coef_df_lasso
#ridge: coef_df_ridge
#lr: lr.coef_
coef_df = pd.DataFrame({
    "Linear": lr.coef_,
    "Ridge": ridge_model.coef_,
    "Lasso": lasso_model.coef_
}, index=x_train.columns)

coef_df
```

	Linear	Ridge	Lasso
USB	-2.536058e-02	-0.025713	-0.025144
Side airbag	6.071532e-18	0.000000	0.000000
Armrest	-7.613931e-03	-0.006806	-0.006632
Previous_Owners	1.804112e-16	0.000000	0.000000
Daytime running lights	-1.493188e-02	-0.015012	-0.014515
...	...	...	...
Type_New	-4.001490e-02	-0.038481	-0.038223
Type_Pre-registered	-7.455820e-02	-0.073167	-0.072800
Type_Used	-1.260528e-01	-0.124349	-0.123844
Paint_Type_Perl effect	7.333024e-03	0.007329	0.007047
Paint_Type_Uni/basic	1.116644e-03	0.001026	0.000709

79 rows × 3 columns

```
#coeff zero => By Lasso
lasso_zero_features = coef_df[coef_df["Lasso"] == 0].index.tolist()
print("Features eliminated by Lasso:", lasso_zero_features)


top_n = 15  # number of features to plot

# get absolute top n features from any model
top_coef_features = coef_df.abs().sum(axis=1).sort_values(ascending=False).head(top_n).index

coef_df.loc[top_coef_features].plot(kind='bar', figsize=(12, 6))
plt.title("Top coefficients comparison (Linear vs Ridge vs Lasso)")
plt.ylabel("Coefficient value")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

```
Features eliminated by Lasso: ['Power steering', 'Bluetooth', 'Driver-side airbag', 'Isofix', 'Radio', 'Side airbag', 'Previous_Owners', 'LED Daytime Running Lights', 'Passenger-side airbag', 'On-board computer', 'Power windows', 'Electrical sic
```



Conclusion & Key Takeaways

What did you notice by performing regularisation?

From the metrics, we notice that not much changed by performing regularization. But the model execution itself is better with Lasson, as features were nullified.

Did the model performance improve? If not, then why?

No, the model performance itself did not improve because an issue with overfitting or coefficients growth was not observed in the base model in the first place.

Did you find overfitting or not?

No, an issue with overfitting was not found.

Was the data sufficent? Is a linear model sufficient?

Yes the data was mostly sufficient as there were no nulls and minimal outliers. Also duplicates were absent. The data itself was mostly clean.

Yes the linear model is mostly sufficient, as test and predicted prices are similar. However, the model can be made better as the error scores are typically high. A more transformative or ensemble or stacking model can be used for better performance.