Predicting Vehicle Prices Using Regression Models

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).

Tasks Overview

The data pipeline for this task involves the following steps:

1. **Dataset Overview**

2. **Data Preprocessing**
3. **Data Visualisation & Exploration**
4. **Model Building**
5. **Regularisation**

1 Data Understanding

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',

Variable	Description
	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

Importing Necessary Libraries

```
# Importing necessary libraries

import pandas as pd
import numpy as np


import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
```

1.1.1

Load the dataset

```
# Load the data
from google.colab import files
uploaded = files.upload()


<IPython.core.display.HTML object>

Saving Car_Price_data.csv to Car_Price_data.csv

import pandas as pd

df = pd.read_csv("Car_Price_data.csv")
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
df.shape

(15915, 23)

df.info()

<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
```

2 Analysis and Feature Engineering [35 marks]

2.1 Preliminary Analysis and Frequency Distributions [13 marks]

2.1.1 [1 marks]

Check and fix missing values.

```
# Find the proportion of missing values in each column and handle if
found
df.isnull().sum().sort_values(ascending=False)
```

```
make_model              0
body_type               0
price                   0
vat                     0
km                      0
Type                    0
Fuel                    0
Gears                   0
Comfort_Convenience     0
Entertainment_Media     0
Extras                  0
Safety_Security         0
age                     0
Previous_Owners         0
hp_kW                   0
Inspection_new          0
Paint_Type              0
Upholstery_type         0
Gearing_Type            0
Displacement_cc         0
Weight_kg               0
Drive_chain             0
```

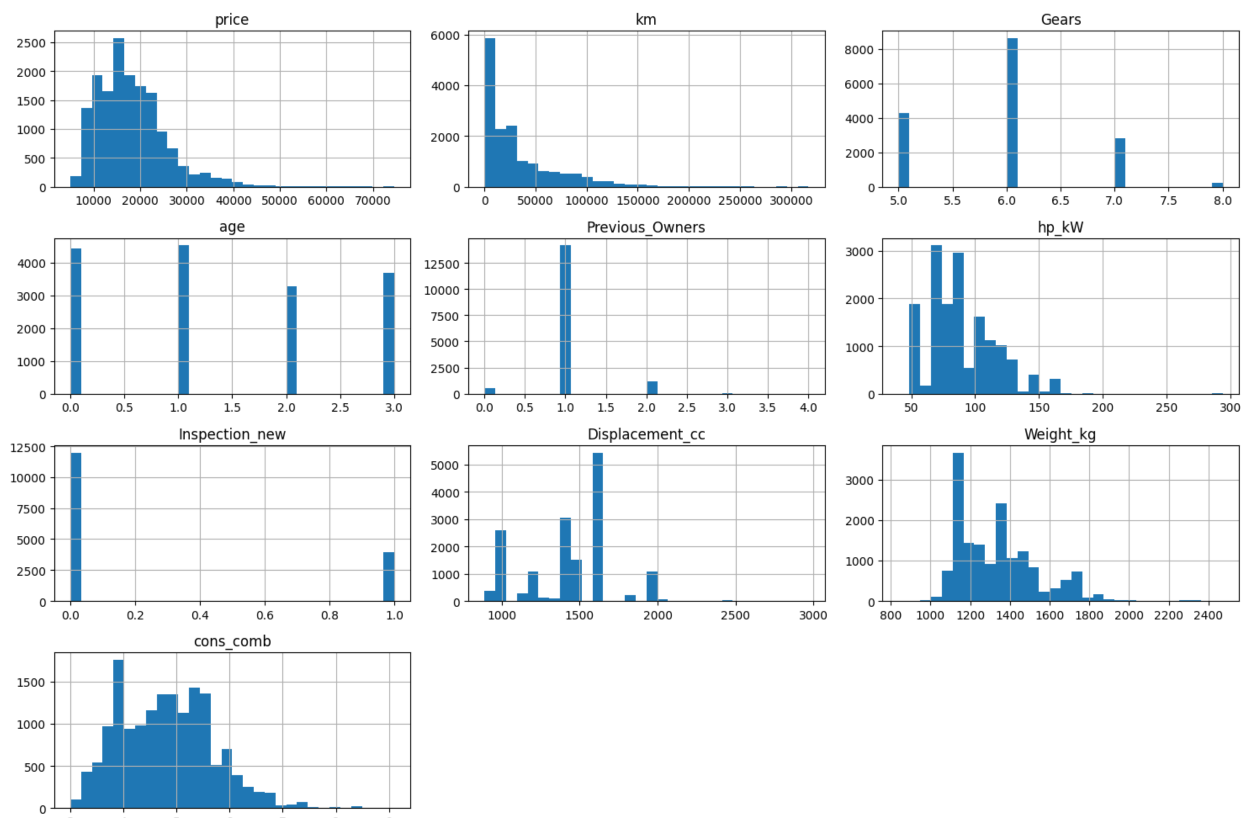
```
cons_comb                0
dtype: int64

# Numerical → median
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Categorical → mode
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

From the features, identify the target feature and numerical and categorical predictors. Select the numerical and categorical features carefully as they will be used in analysis.

2.1.2 [3 marks]

Identify numerical predictors and plot their frequency distributions.

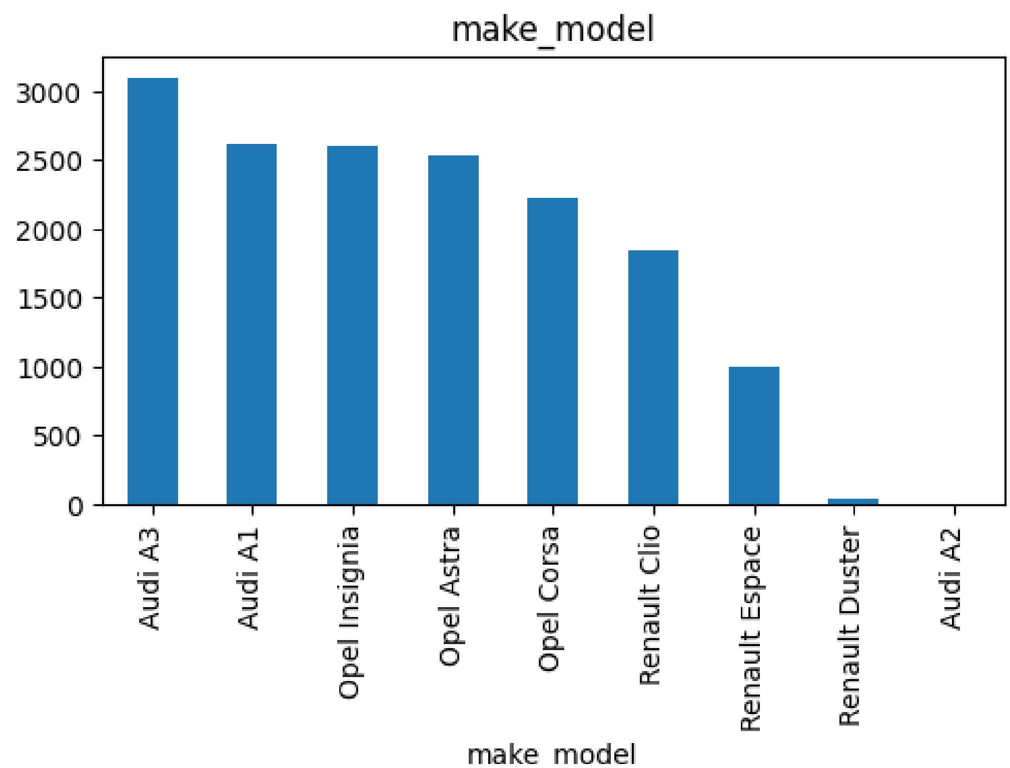
```
# Identify numerical features and plot histograms
df[num_cols].hist(figsize=(15,10), bins=30)
plt.tight_layout()
plt.show()
```

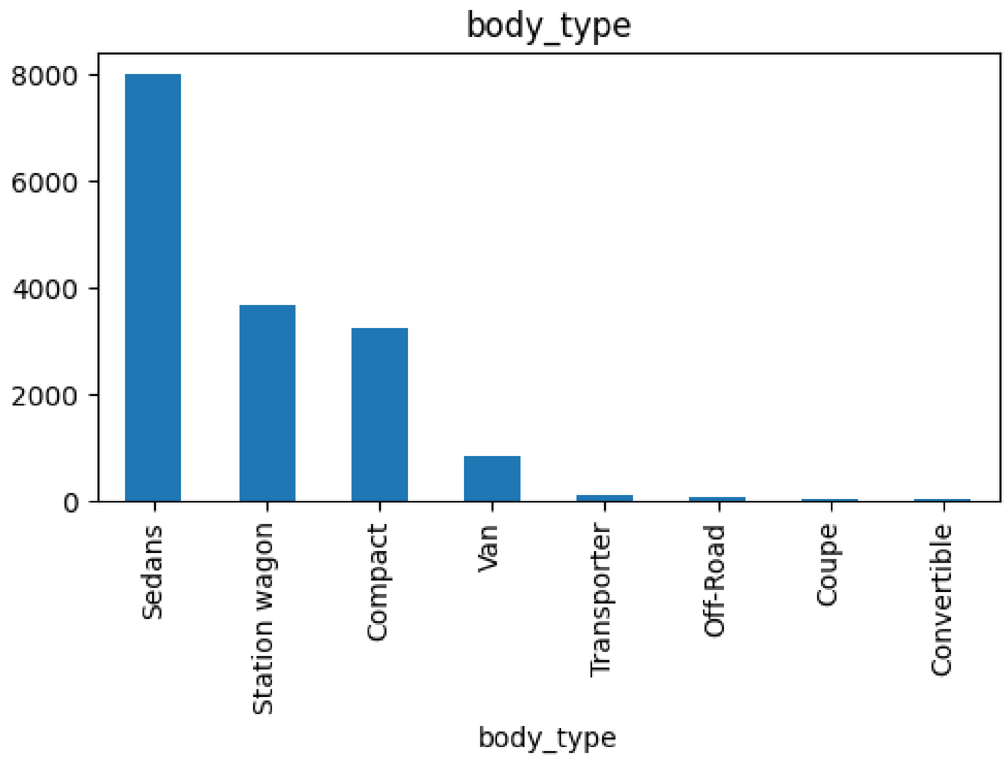


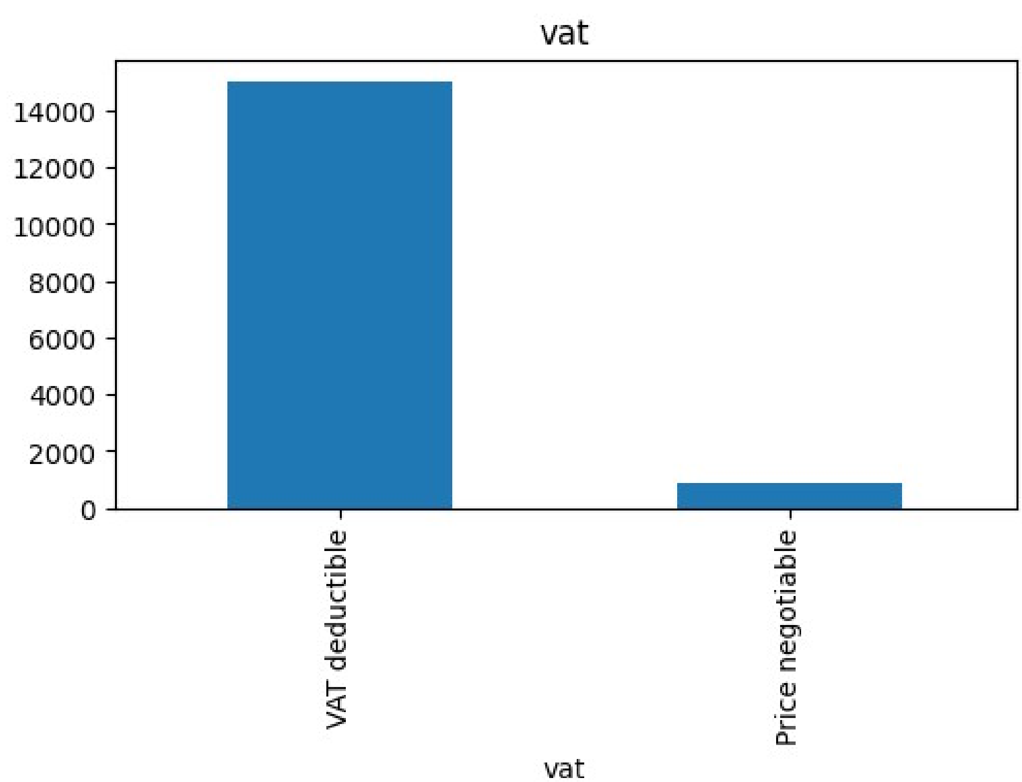
2.1.3 [3 marks]

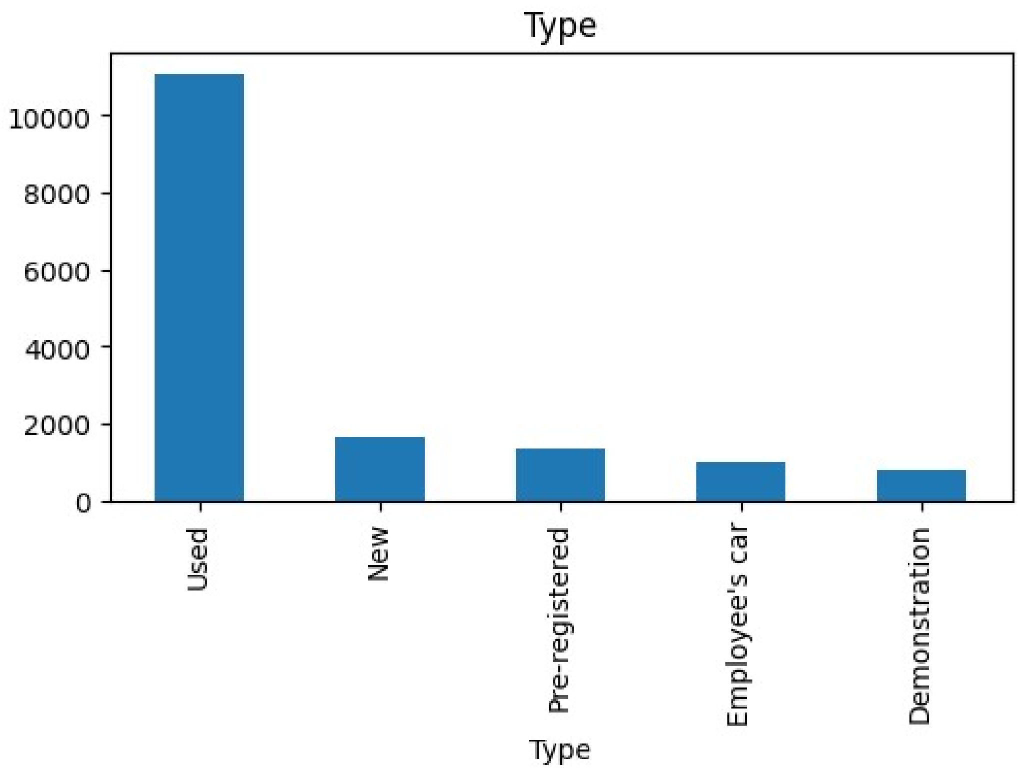
Identify categorical predictors and plot their frequency distributions.

```
# Identify categorical columns and check their frequency distributions
for col in cat_cols:
    plt.figure(figsize=(6,3))
    df[col].value_counts().head(10).plot(kind='bar')
    plt.title(col)
    plt.show()
```

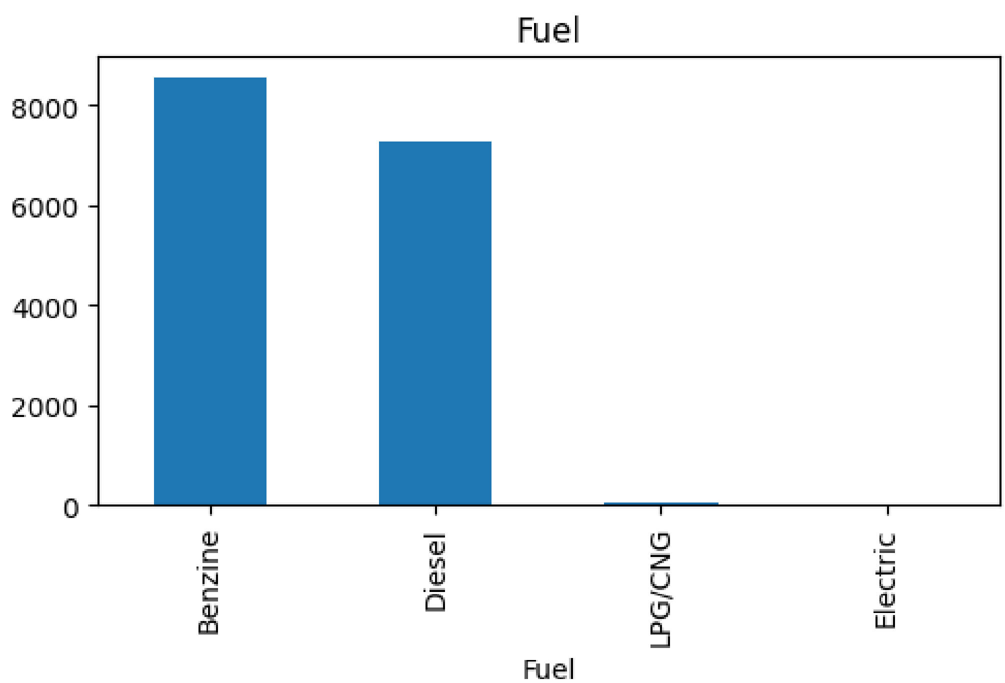



body_type
8000
6000
4000
2000
0
Sedans
Station wagon
Compact
Van
Transporter
Off-Road
Coupe
Convertible
body_type

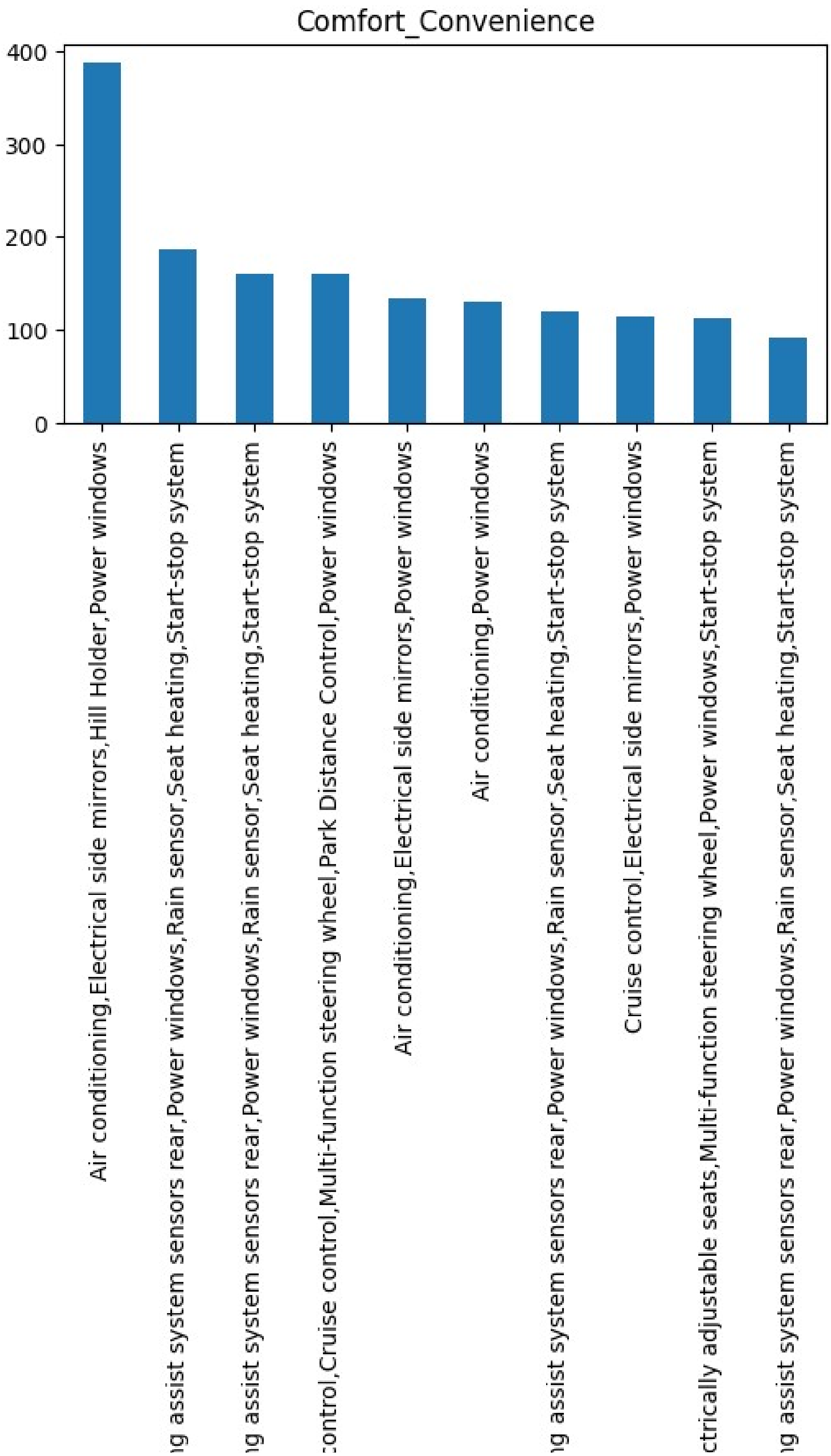

vat
14000
12000
10000
8000
6000
4000
2000
0
VAT deductible
Price negotiable
vat

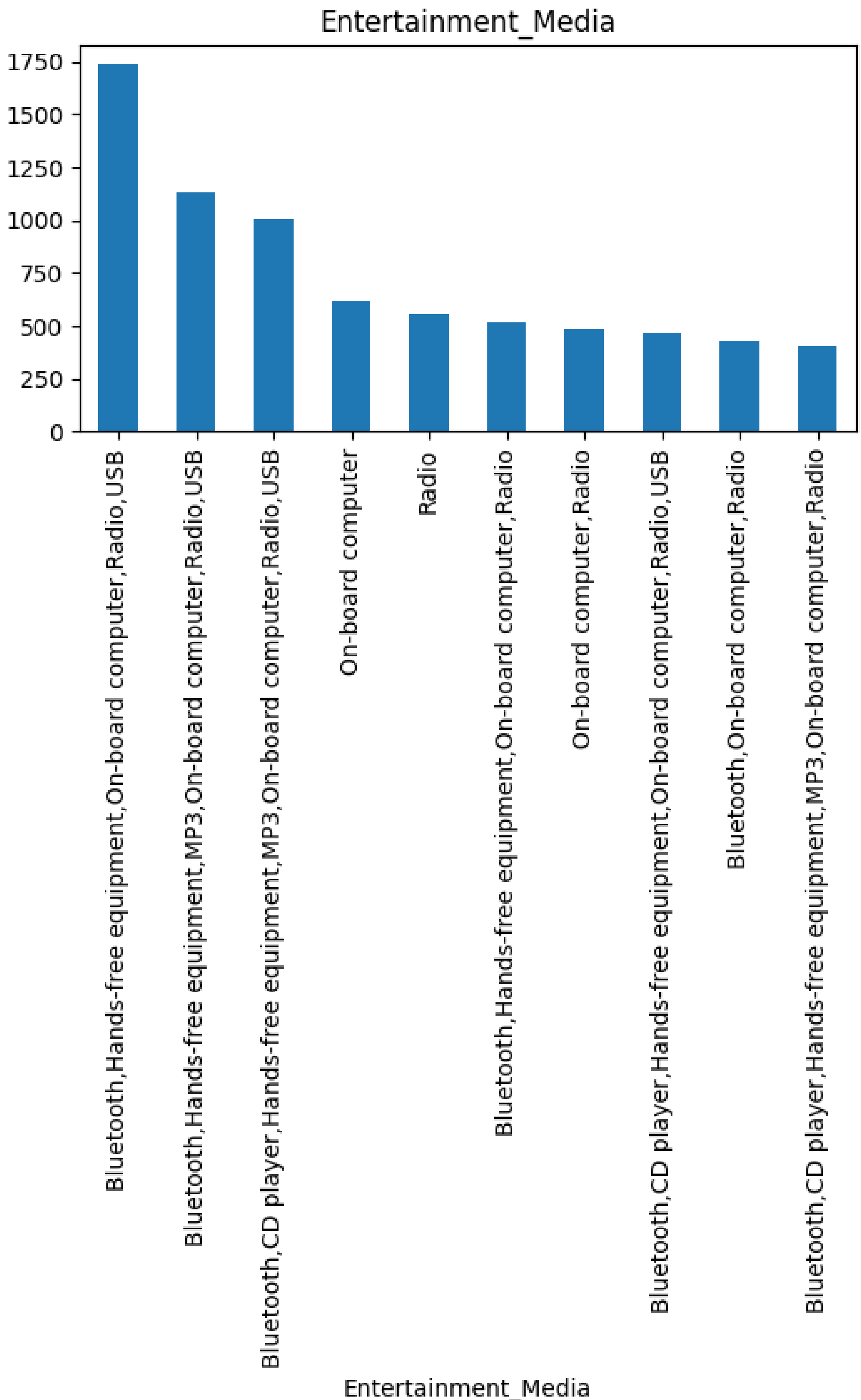


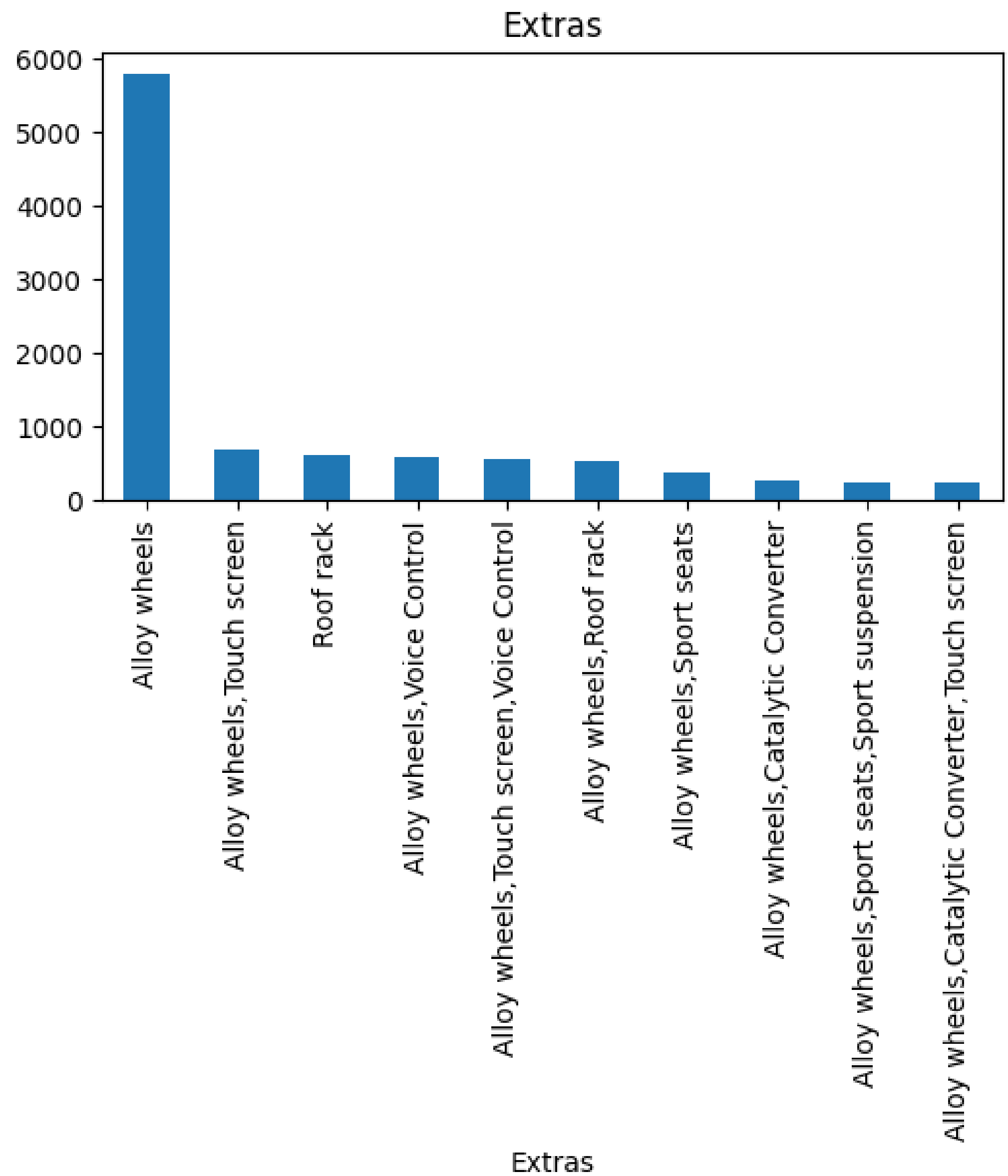
Type
10000
8000
6000
4000
2000
0
Used
New
Pre-registered
Employee's car
Demonstration
Type

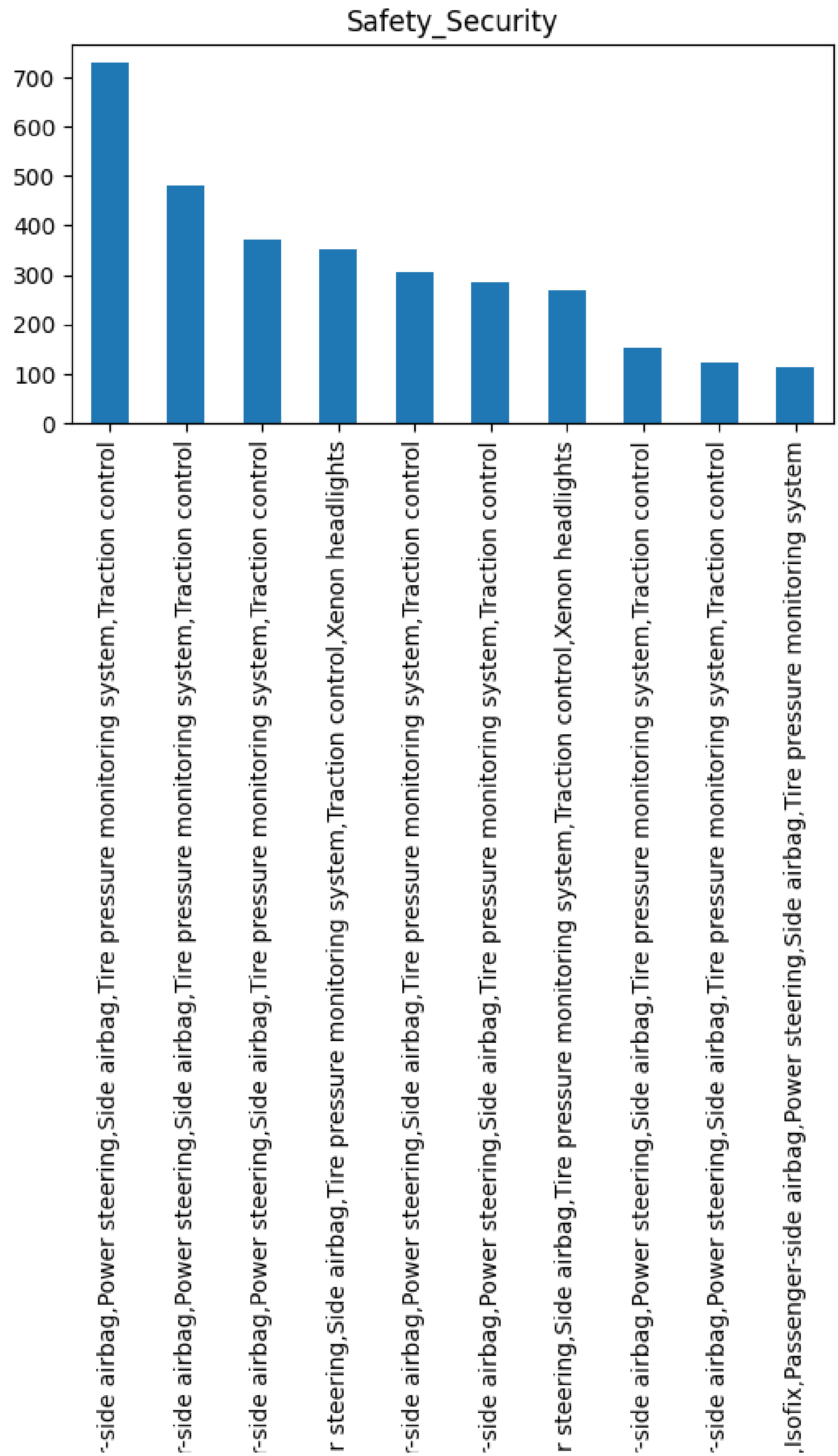


Fuel
8000
6000
4000
2000
0
Benzine
Diesel
LPG/CNG
Electric
Fuel

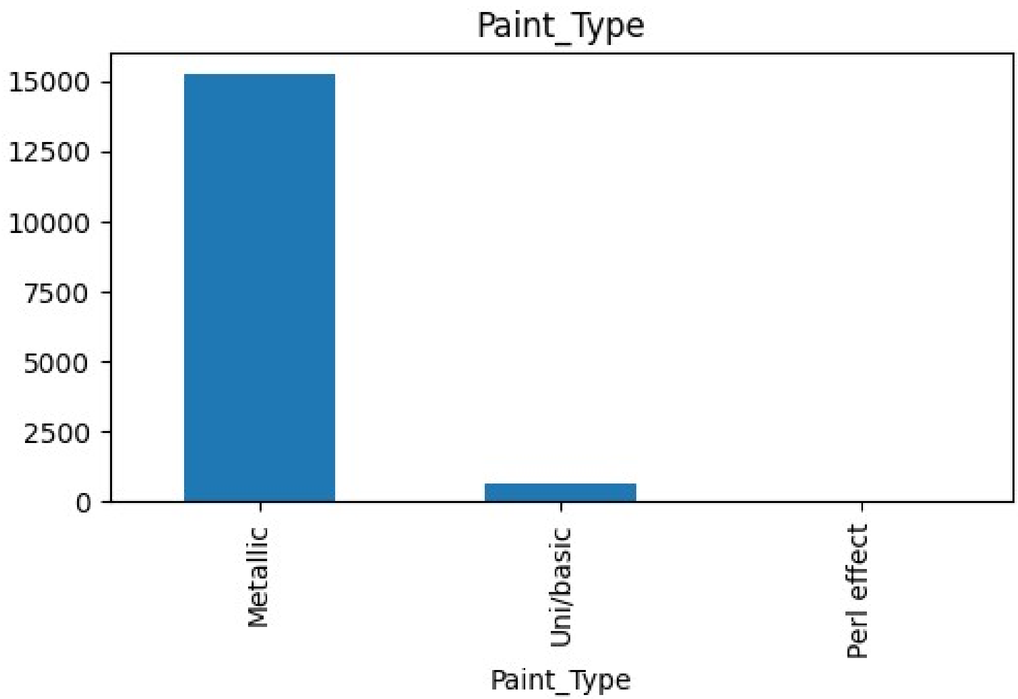

Comfort_Convenience
400
300
200
100
0
Air conditioning,Electrical side mirrors,Hill Holder,Power windows
ng assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system
ng assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system
control,Cruise control,Multi-function steering wheel,Park Distance Control,Power windows
Air conditioning,Electrical side mirrors,Power windows
Air conditioning,Power windows
ng assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system
Cruise control,Electrical side mirrors,Power windows
ctrically adjustable seats,Multi-function steering wheel,Power windows,Start-stop system
ng assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system

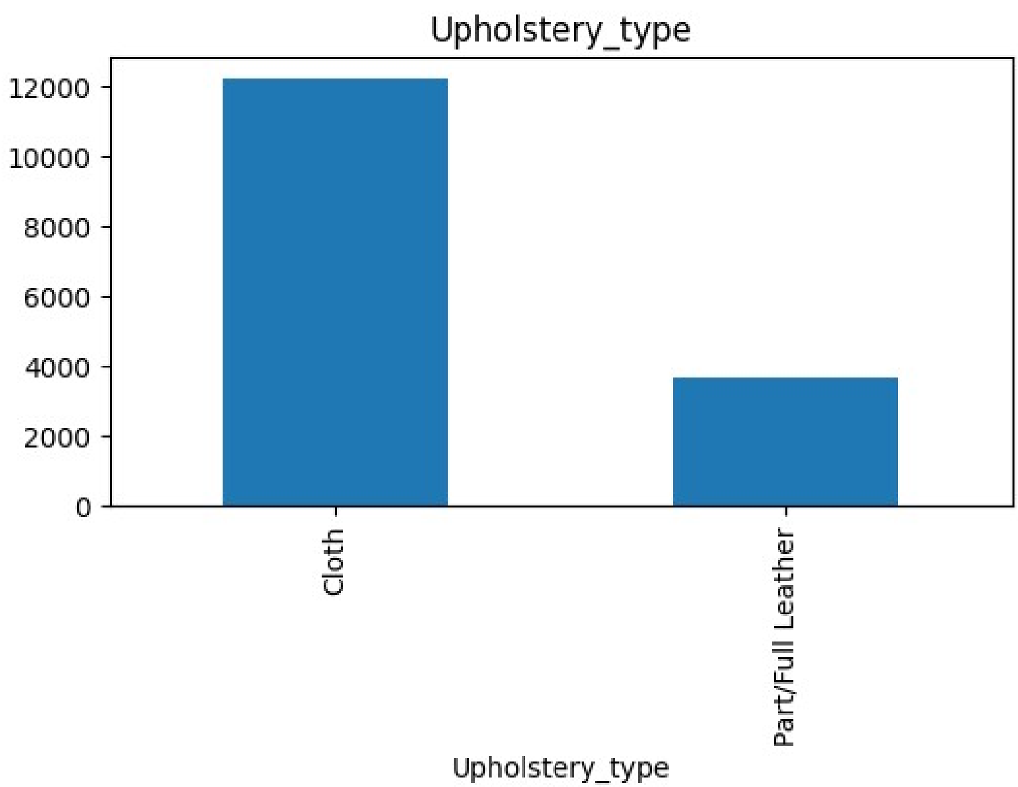

Entertainment_Media
1750
1500
1250
1000
750
500
250
0
Bluetooth,Hands-free equipment,On-board computer,Radio,USB
Bluetooth,Hands-free equipment,MP3,On-board computer,Radio,USB
Bluetooth,CD player,Hands-free equipment,MP3,On-board computer,Radio,USB
On-board computer
Radio
Bluetooth,Hands-free equipment,On-board computer,Radio
On-board computer,Radio
Bluetooth,CD player,Hands-free equipment,On-board computer,Radio,USB
Bluetooth,On-board computer,Radio
Bluetooth,CD player,Hands-free equipment,MP3,On-board computer,Radio
Entertainment_Media

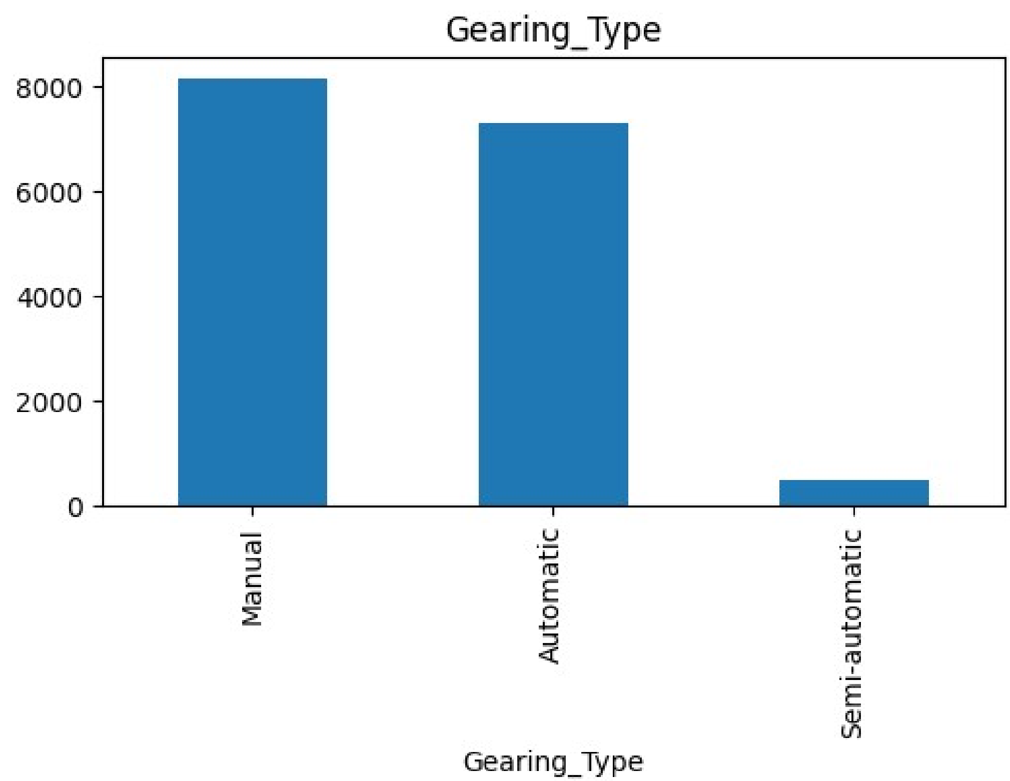

Extras
6000
5000
4000
3000
2000
1000
0
Alloy wheels
Alloy wheels,Touch screen
Roof rack
Alloy wheels,Voice Control
Alloy wheels,Touch screen,Voice Control
Alloy wheels,Roof rack
Alloy wheels,Sport seats
Alloy wheels,Catalytic Converter
Alloy wheels,Sport seats,Sport suspension
Alloy wheels,Catalytic Converter,Touch screen
Extras

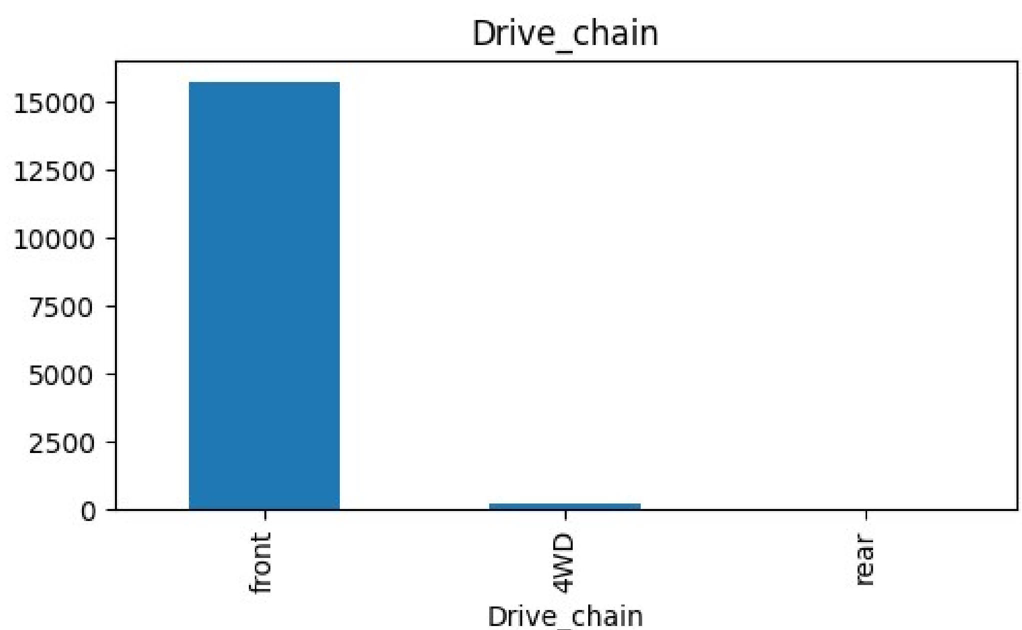


Safety_Security
0
100
200
300
400
500
600
700
-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
r-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
r-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
r steering,Side airbag,Tire pressure monitoring system,Traction control,Xenon headlights
r-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
r steering,Side airbag,Tire pressure monitoring system,Traction control,Xenon headlights
-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
,Isofix,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system


Paint_Type
15000
12500
10000
7500
5000
2500
0
Metallic
Uni/basic
Perl effect
Paint_Type


Upholstery_type
12000
10000
8000
6000
4000
2000
0
Cloth
Part/Full Leather
Upholstery_type





Note: Look carefully at the values stored in columns `["Comfort_Convenience", "Entertainment_Media", "Extras", "Safety_Security"]`.

Should they be considered categorical? Should they be dropped or handled any other way?

2.1.4 [3 marks]

Fix columns with low frequency values and class imbalances.

Some information regarding values in the `Type` column that may help:

- *'Pre-registered'* cars are ones which have already been registered previously by the seller.
- *'New'* cars are not necessarily new cars, but new-like cars. These might also have multiple owners due to multiple pre-registrations as well.
- *'Employee's car'* are cars used by employees over a short period of time and small distance.
- *'Demonstration'* cars are used for trial purposes and also driven for a short time and distance.

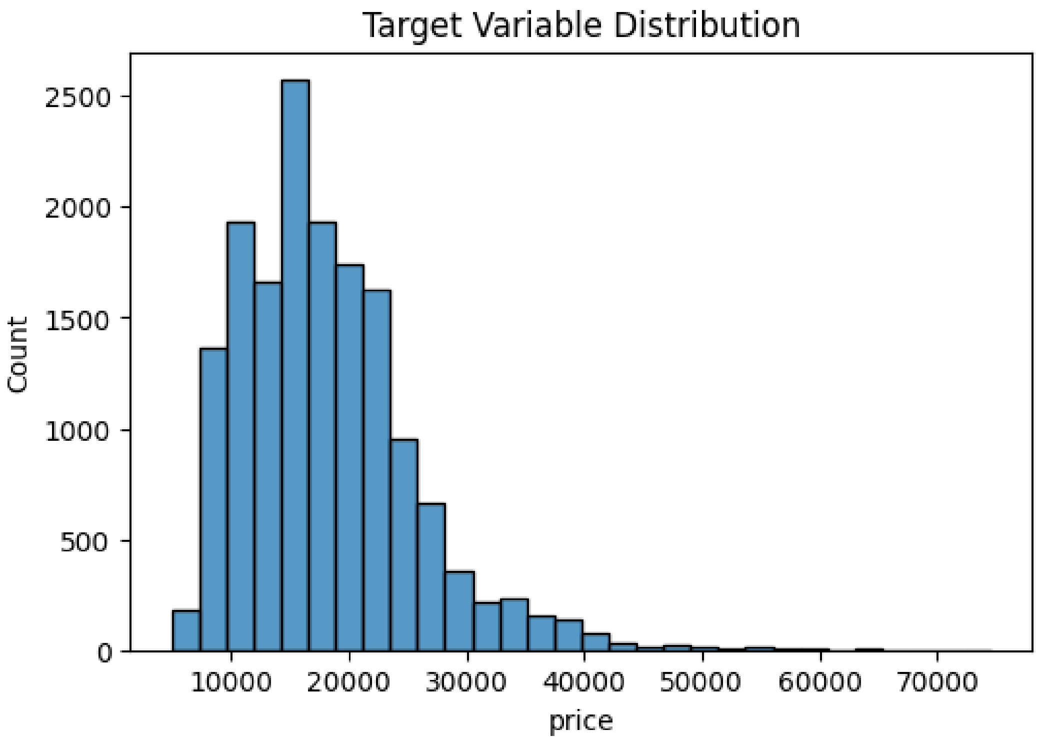
Based on these, you can handle this particular column. For other columns, decide a strategy on your own.

```
# Fix columns as needed
def group_rare(df, col, threshold=0.01):
    freq = df[col].value_counts(normalize=True)
    rare = freq[freq < threshold].index
    df[col] = df[col].replace(rare, "Other")
    return df

for col in cat_cols:
    df = group_rare(df, col)
```

2.1.5 [3 marks]

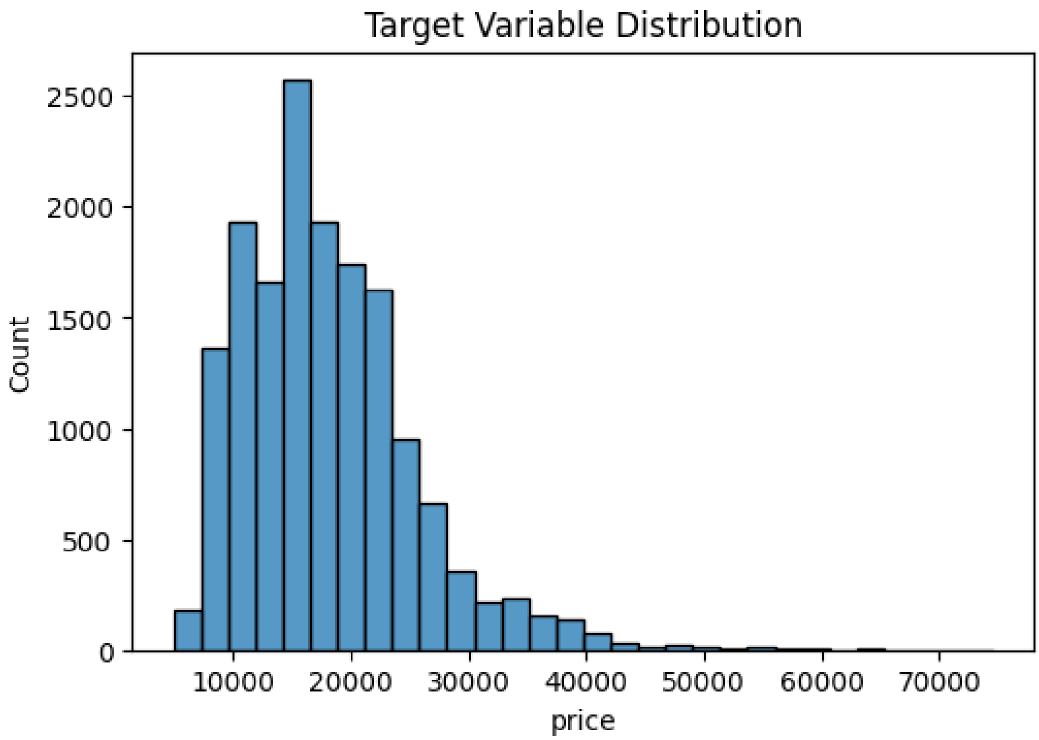
Identify target variable and plot the frequency distributions. Apply necessary transformations.

```
# Plot histograms for target feature
target = "price"   # adjust if column name differs

plt.figure(figsize=(6,4))
sns.histplot(df[target], bins=30)
plt.title("Target Variable Distribution")
plt.show()
```



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

```
# Transform the target feature

target = "price"   # adjust if column name differs

plt.figure(figsize=(6,4))
sns.histplot(df[target], bins=30)
plt.title("Target Variable Distribution")
plt.show()
```

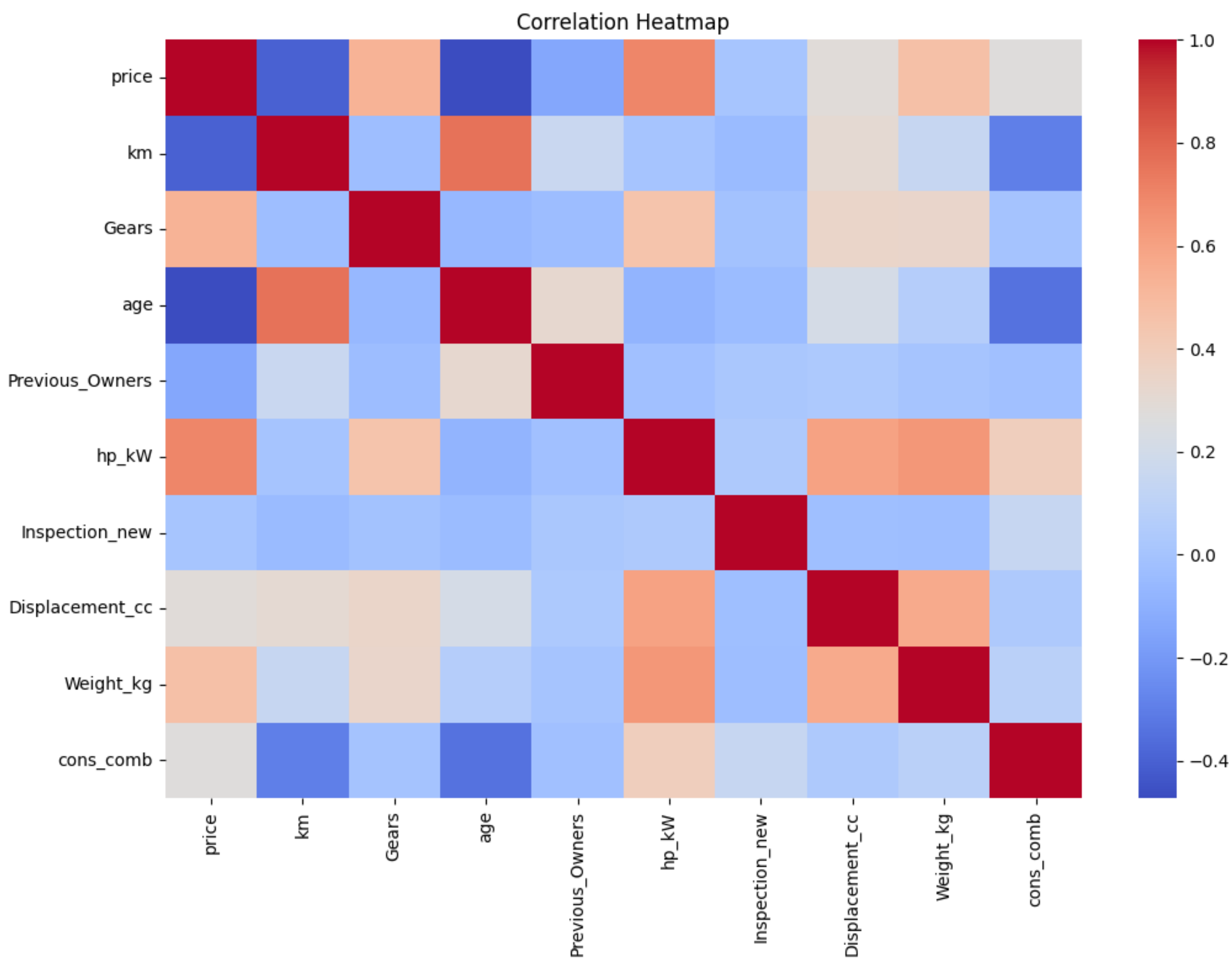


2.2 Correlation analysis [6 marks]

2.2.1 [3 marks]

Plot the correlation map between features and target variable.

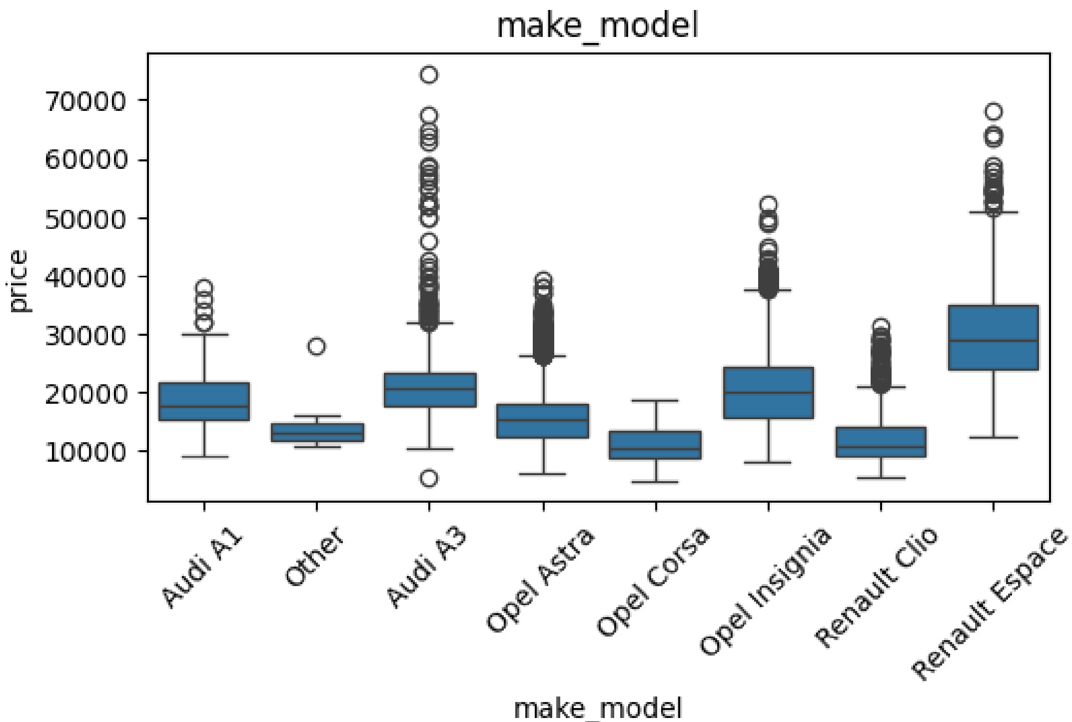
```
# Visualise correlation
plt.figure(figsize=(12,8))
sns.heatmap(df[num_cols].corr(), cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

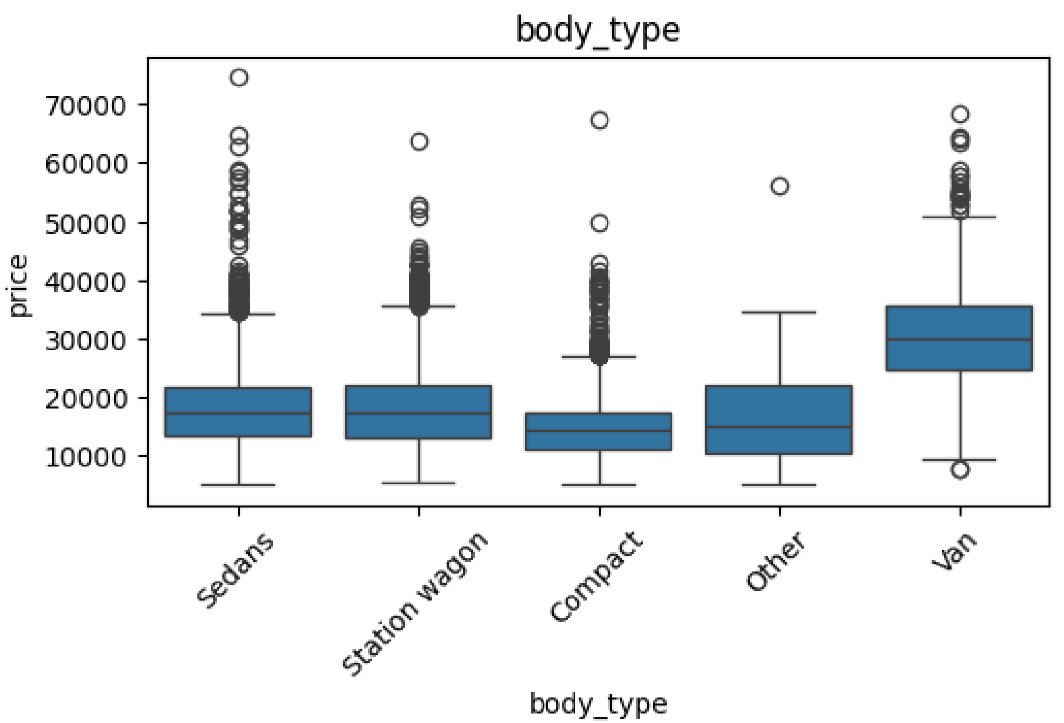


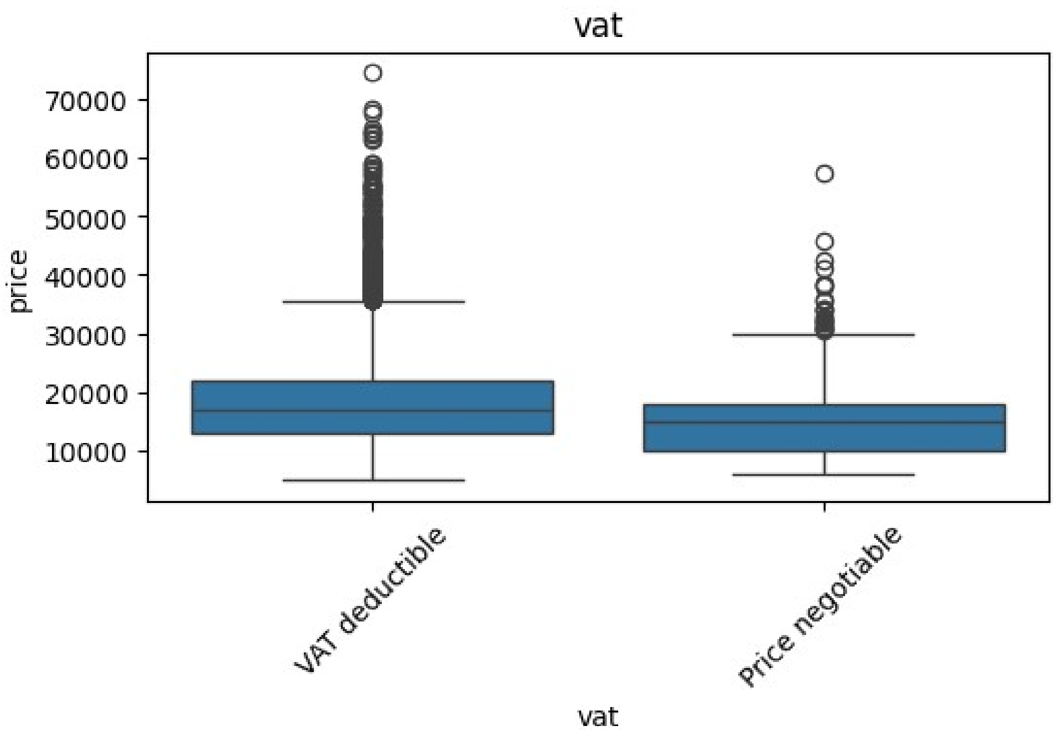
2.2.2 [3 marks]

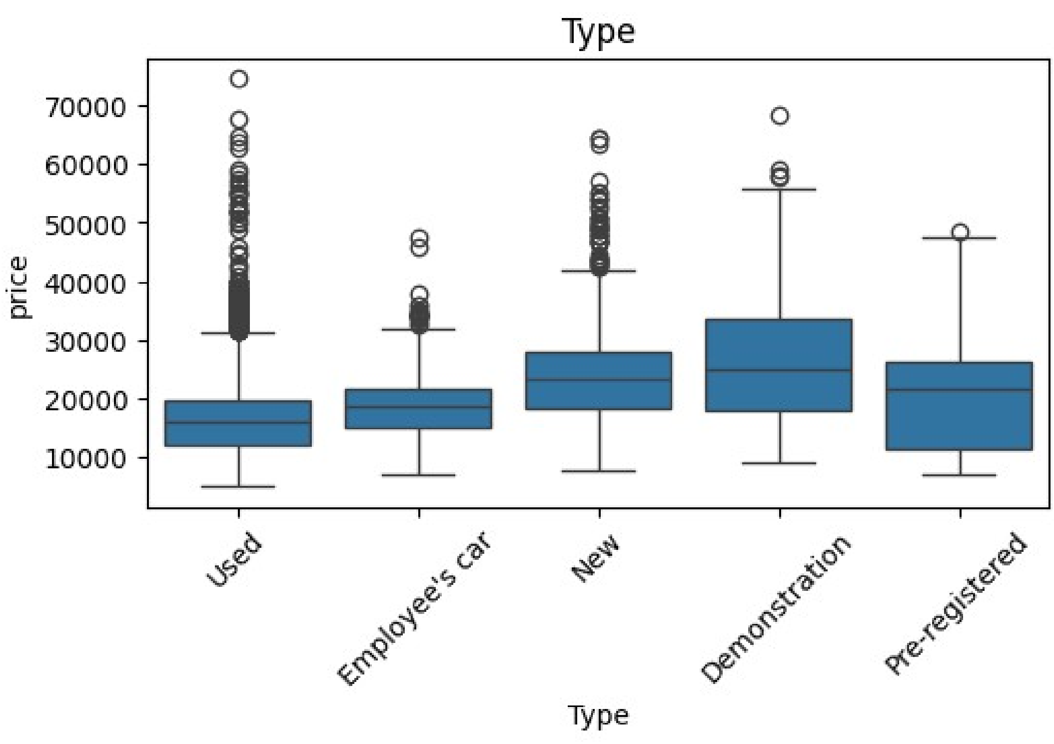
Analyse correlation between categorical features and target variable.

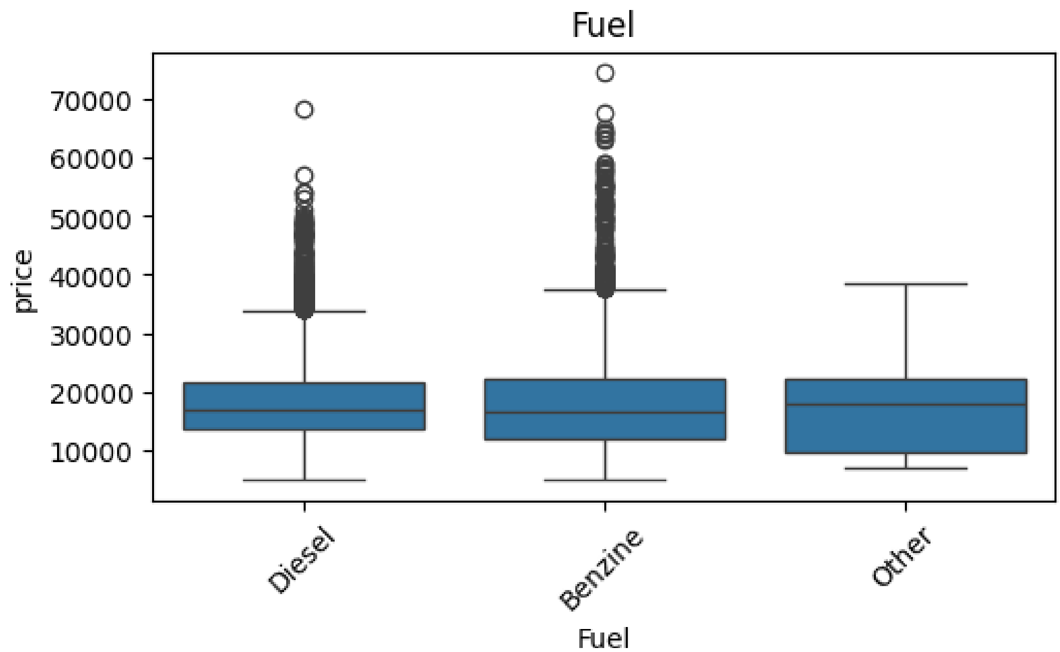
```
# Comparing average values of target for different categories

for col in cat_cols:
    plt.figure(figsize=(6,3))
    sns.boxplot(x=df[col], y=df[target])
    plt.xticks(rotation=45)
    plt.title(col)
    plt.show()
```


make_model
price
70000
60000
50000
40000
30000
20000
10000
Audi A1
Other
Audi A3
Opel Astra
Opel Corsa
Opel Insignia
Renault Clio
Renault Espace
make_model

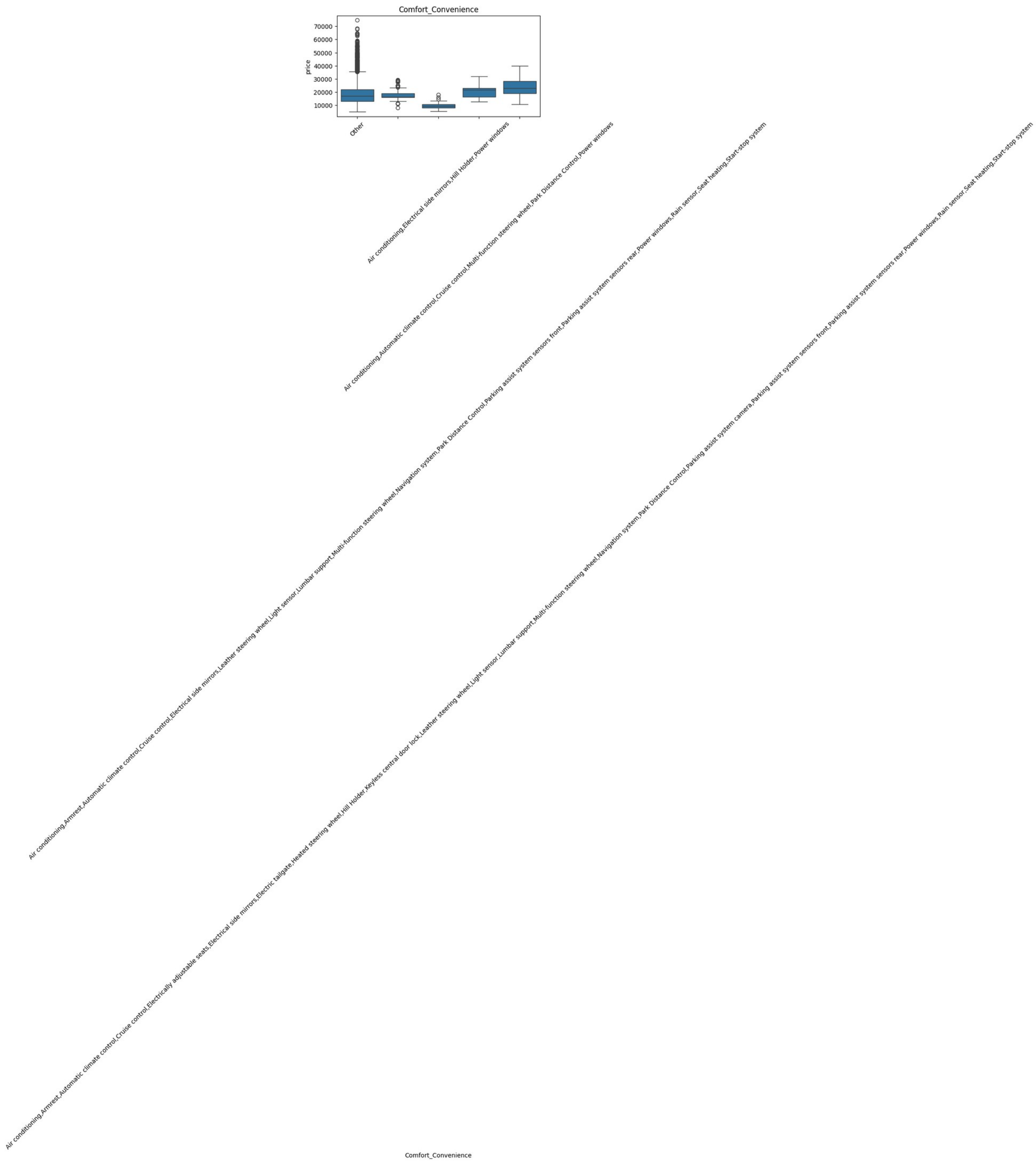

body_type
price
70000
60000
50000
40000
30000
20000
10000
Sedans
Station wagon
Compact
Other
Van
body_type

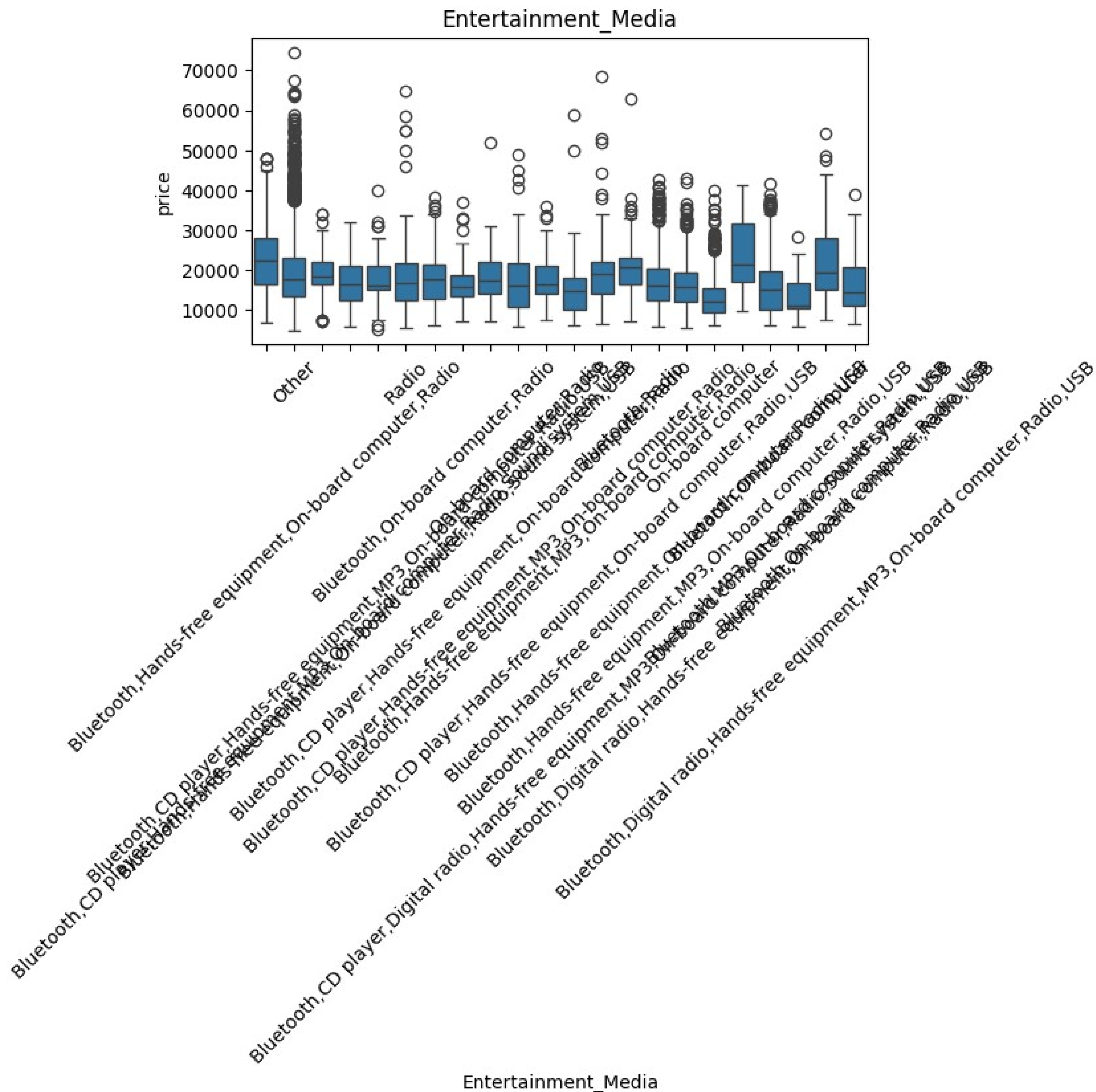

vat
70000
60000
50000
40000
30000
20000
10000
price
VAT deductible
Price negotiable
vat

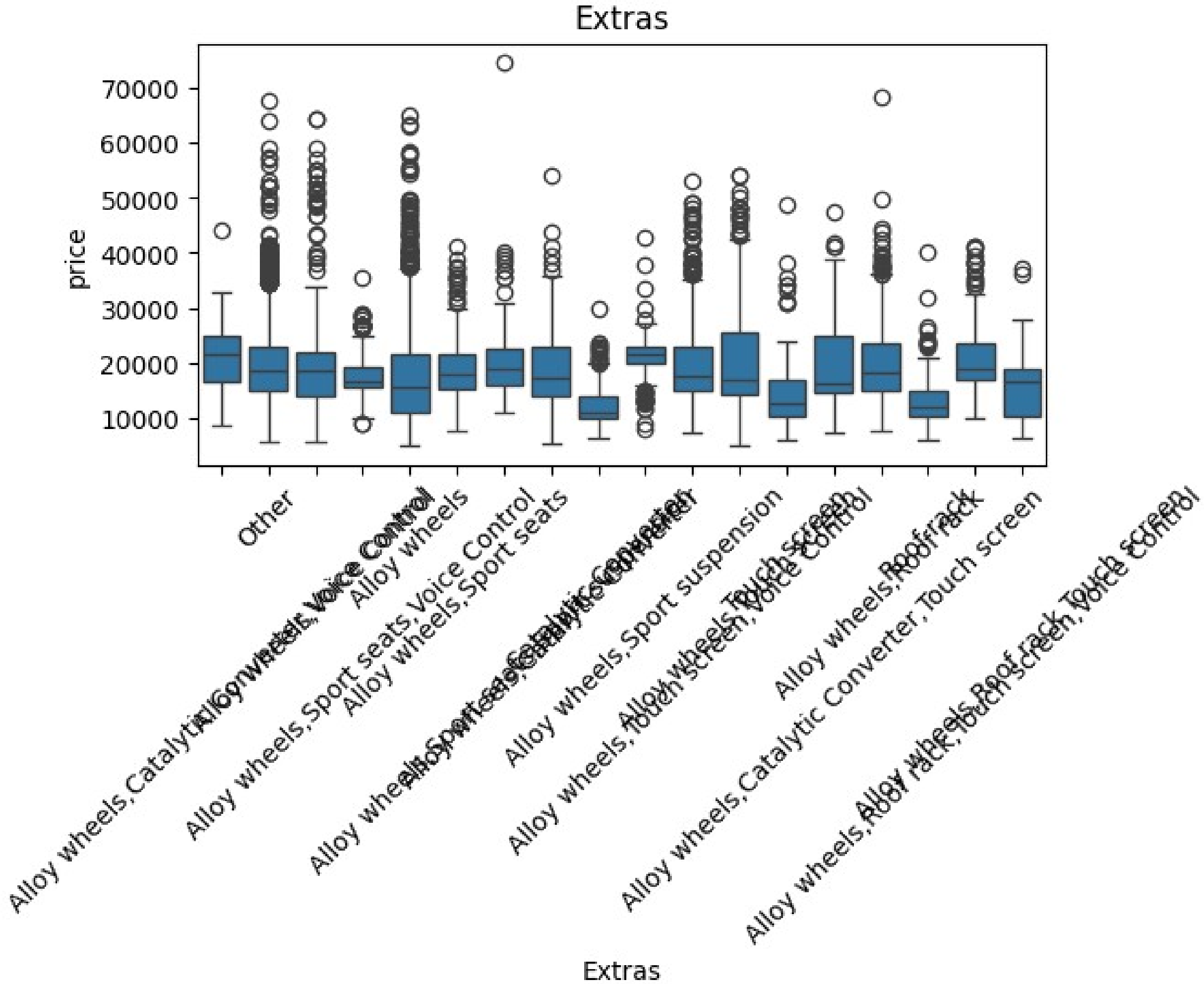

Type
70000
60000
50000
40000
30000
20000
10000
price
Used
Employee's car
New
Demonstration
Pre-registered
Type

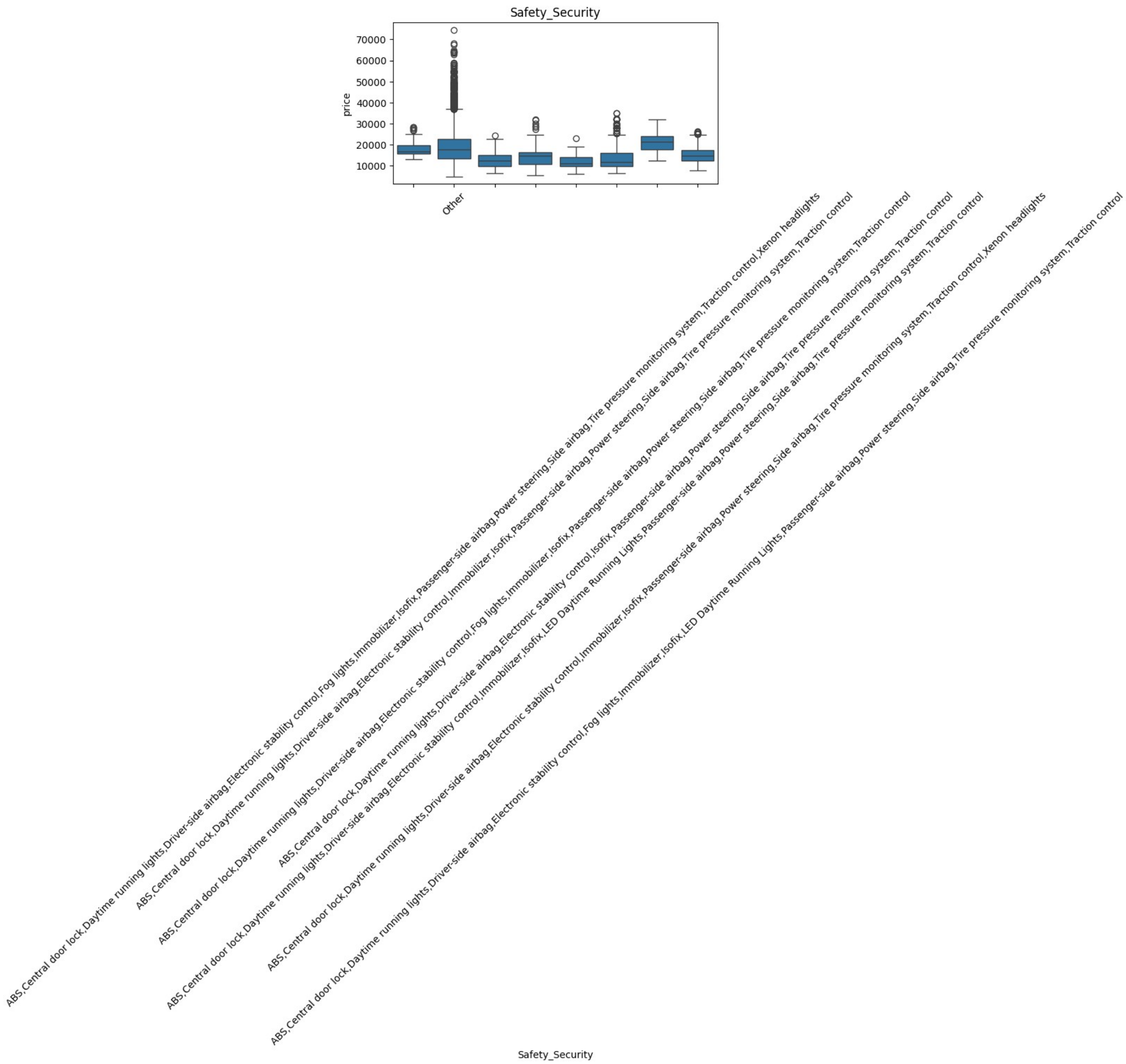


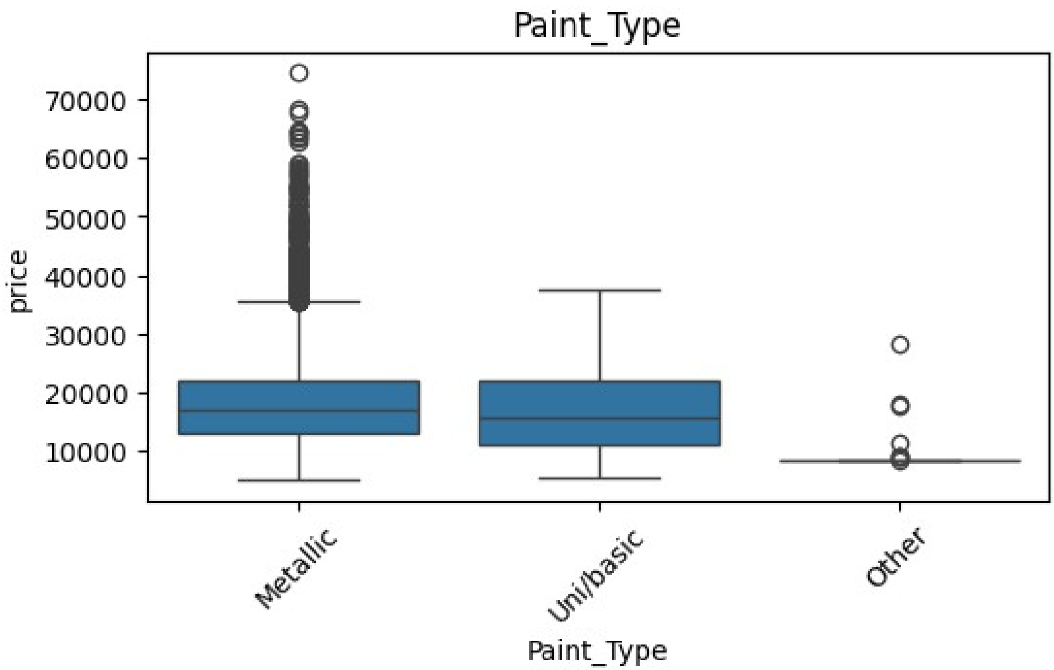
Fuel
70000
60000
50000
40000
30000
20000
10000
price
Diesel
Benzine
Other
Fuel


Comfort_Convenience
70000
60000
50000
40000
30000
20000
10000
price
Other
Air conditioning,Electrical side mirrors,Hill Holder,Power windows
Air conditioning,Automatic climate control,Cruise control,Multi-function steering wheel,Park Distance Control,Power windows
Air conditioning,Armrest,Automatic climate control,Cruise control,Electrical side mirrors,Leather steering wheel,Light sensor,Lumbar support,Multi-function steering wheel,Navigation system,Park Distance Control,Parking assist system sensors front,Parking assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system
Air conditioning,Armrest,Automatic climate control,Cruise control,Electrically adjustable seats,Electrical side mirrors,Electric tailgate,Heated steering wheel,Hill Holder,Keyless central door lock,Leather steering wheel,Light sensor,Lumbar support,Multi-function steering wheel,Navigation system,Park Distance Control,Parking assist system camera,Parking assist system sensors front,Parking assist system sensors rear,Power windows,Rain sensor,Seat heating,Start-stop system
Comfort_Convenience

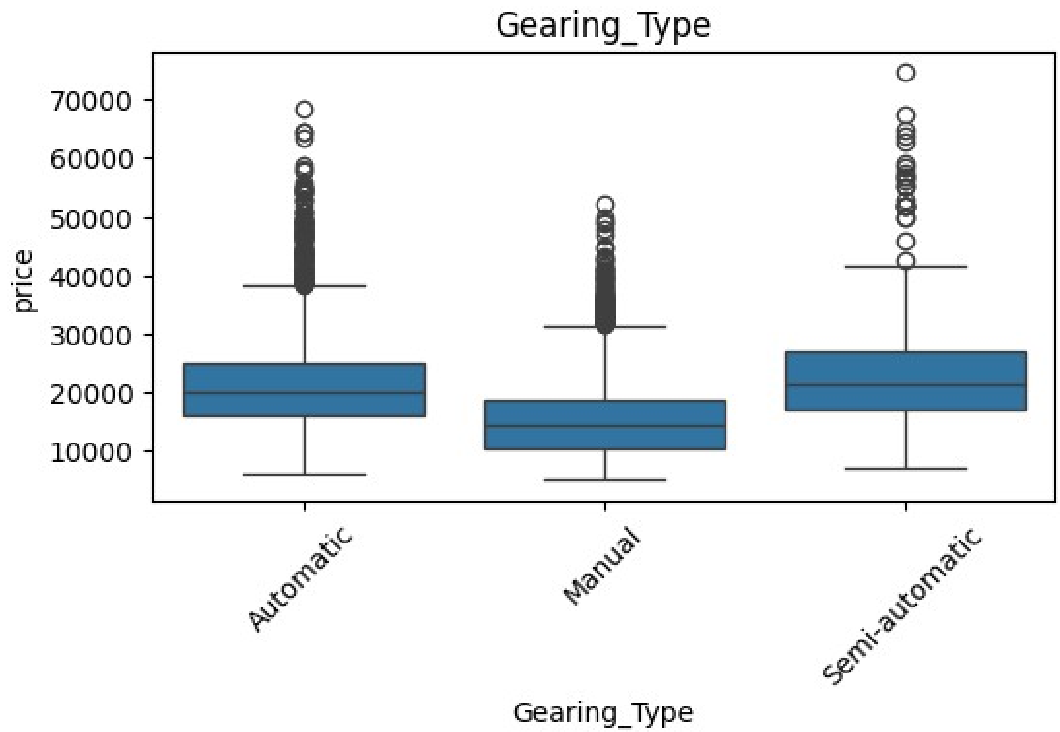

Entertainment_Media
price
70000
60000
50000
40000
30000
20000
10000
Other
Radio
Bluetooth,Hands-free equipment,On-board computer,Radio
Bluetooth,On-board computer,Radio
Bluetooth,Digital radio,Hands-free equipment,MP3,On-board computer,Radio,USB
Entertainment_Media

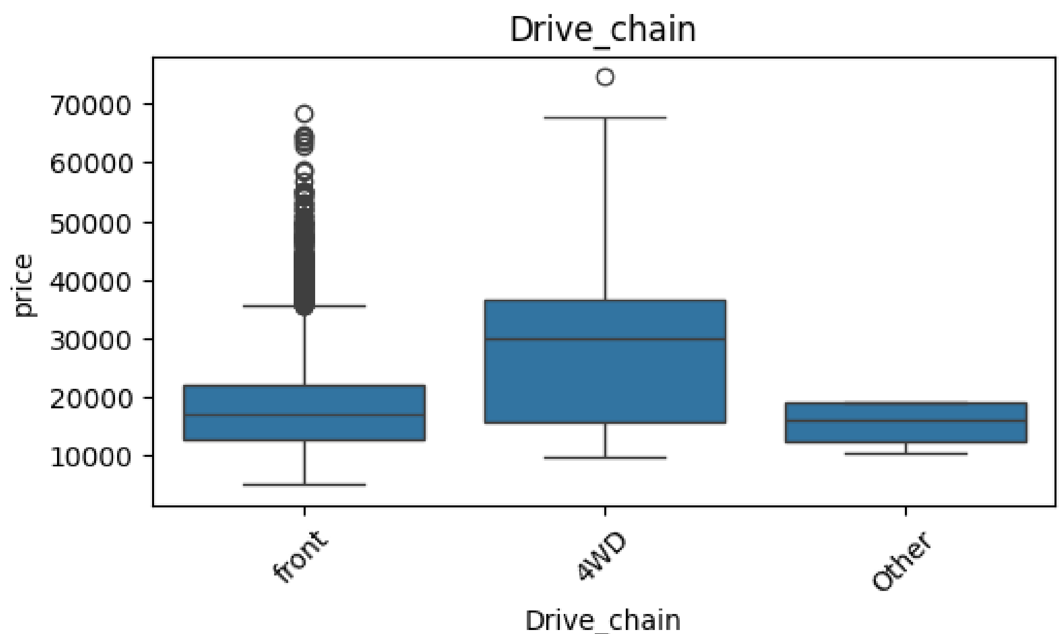

Extras
price
70000
60000
50000
40000
30000
20000
10000
Other
Alloy wheels
Alloy wheels,Sport seats,Voice Control
Alloy wheels,Sport seats
Alloy wheels,Sport suspension
Alloy wheels,Roof rack
Alloy wheels,Catalytic Converter,Touch screen
Extras


Safety_Security
price
70000
60000
50000
40000
30000
20000
10000
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
Other
ABS,Central door lock,Daytime running lights,Driver-side airbag,Electronic stability control,Immobilizer,Isofix,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
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
ABS,Central door lock,Daytime running lights,Driver-side airbag,Electronic stability control,Isofix,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
ABS,Central door lock,Daytime running lights,Driver-side airbag,Electronic stability control,Immobilizer,Isofix,LED Daytime Running Lights,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
ABS,Central door lock,Daytime running lights,Driver-side airbag,Electronic stability control,Immobilizer,Isofix,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control,Xenon headlights
ABS,Central door lock,Daytime running lights,Driver-side airbag,Electronic stability control,Fog lights,Immobilizer,Isofix,LED Daytime Running Lights,Passenger-side airbag,Power steering,Side airbag,Tire pressure monitoring system,Traction control
Safety_Security


Paint_Type
70000
60000
50000
40000
30000
20000
10000
price
Metallic
Uni/basic
Other
Paint_Type


Upholstery_type
70000
60000
50000
40000
30000
20000
10000
price
Cloth
Part/Full Leather
Upholstery_type





2.3 Outlier analysis [5 marks]

2.3.1 [2 marks]

Identify potential outliers in the data.

```
# Outliers present in each column

Q1 = df[target].quantile(0.25)
Q3 = df[target].quantile(0.75)
IQR = Q3 - Q1

lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
```

2.3.2 [3 marks]

Handle the outliers suitably.

```
# Handle outliers
df[target] = np.where(df[target] > upper, upper, df[target])
df[target] = np.where(df[target] < lower, lower, df[target])
```

2.4 Feature Engineering [11 marks]

2.4.1

Fix any redundant columns and create new ones if needed.

```
# Fix/create columns as needed
spec_list_cols = [
    'Comfort_Convenience',
    'Entertainment_Media',
    'Extras',
    'Safety_Security'
]

for col in spec_list_cols:
    df[col] = df[col].apply(
        lambda x: len(str(x).split(',')) if pd.notnull(x) else 0
    )

low_variance_cols = []

for col in df.columns:
    if df[col].nunique() <= 1:
        low_variance_cols.append(col)

df.drop(columns=low_variance_cols, inplace=True)
```

2.4.2 [4 marks]

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.

```
# Check unique values in each feature spec column
df_encoded = pd.get_dummies(df, drop_first=True)
```

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

X = df_encoded.drop(columns=[target])
y = df_encoded[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

2.4.3 [3 marks]

Perform feature encoding.

```
# Encode features

df_encoded = pd.get_dummies(df, drop_first=True)
```

2.4.4 [2 marks]

Split the data into training and testing sets.

```
# Split data
target = 'price'   # confirm column name

X = df_encoded.drop(columns=[target])
y = df_encoded[target]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
```

2.4.5 [2 marks]

Scale the features.

```python
# Scale features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3 Linear Regression Models [35 marks]

3.1 Baseline Linear Regression Model [10 marks]

3.1.1 [5 marks]

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

```python
# Initialise and train model
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

y_pred_lr = lr.predict(X_test_scaled)

# Evaluate the model's performance
print("R2:", r2_score(y_test, y_pred_lr))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_lr)))
print("MAE:", mean_absolute_error(y_test, y_pred_lr))
```

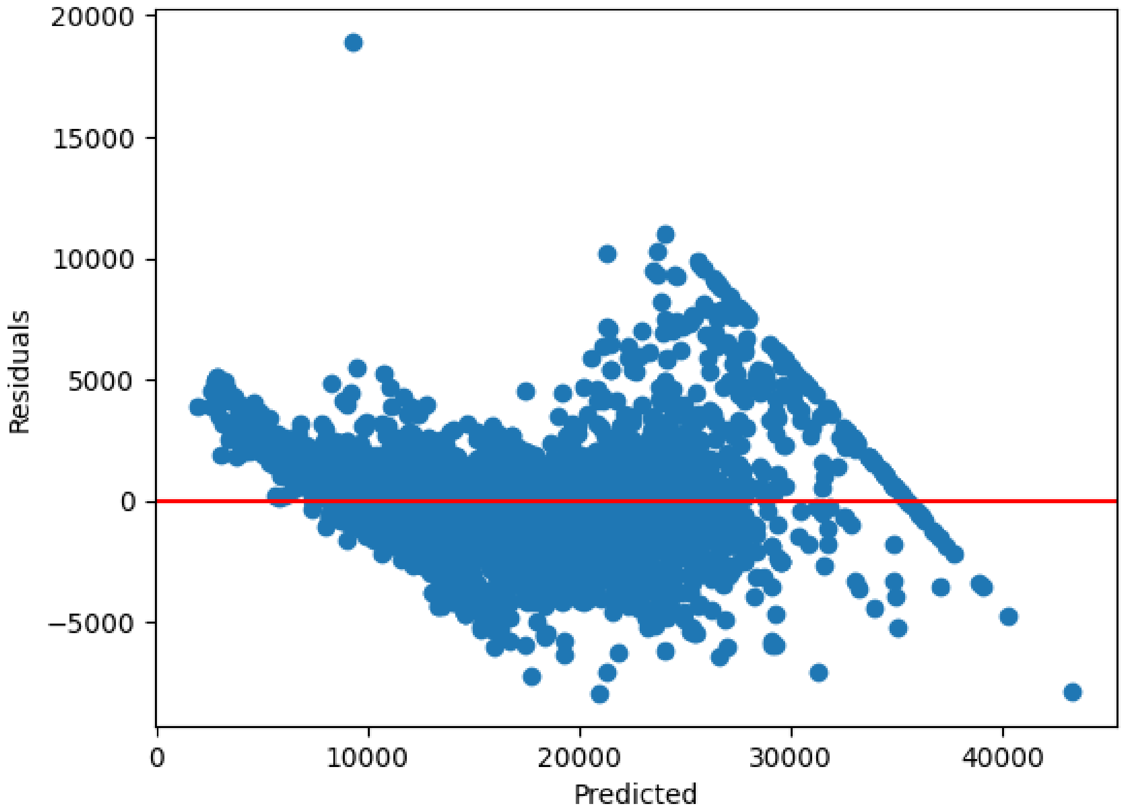
```
R2: 0.8861268215510806
RMSE: 2301.816612474634
MAE: 1675.265187214249
```

3.1.2 [5 marks]

Analyse residuals and check other assumptions of linear regression.

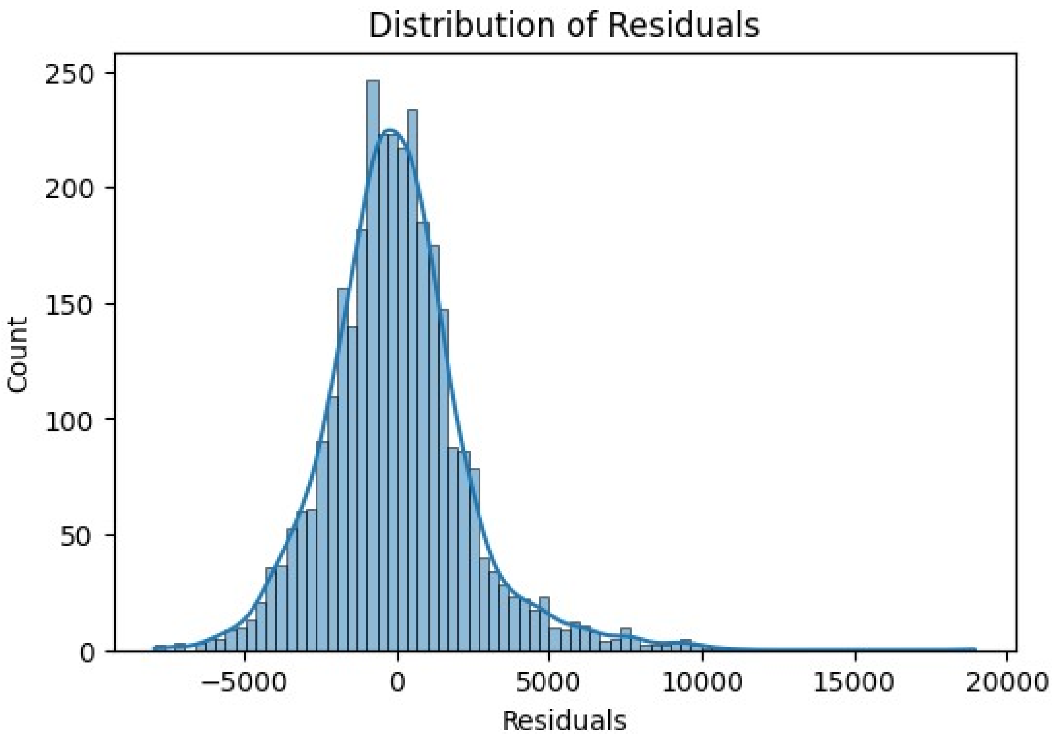
Check for linearity by analysing residuals vs predicted values

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

plt.scatter(y_pred_lr, residuals)
plt.axhline(0, color='red')
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```

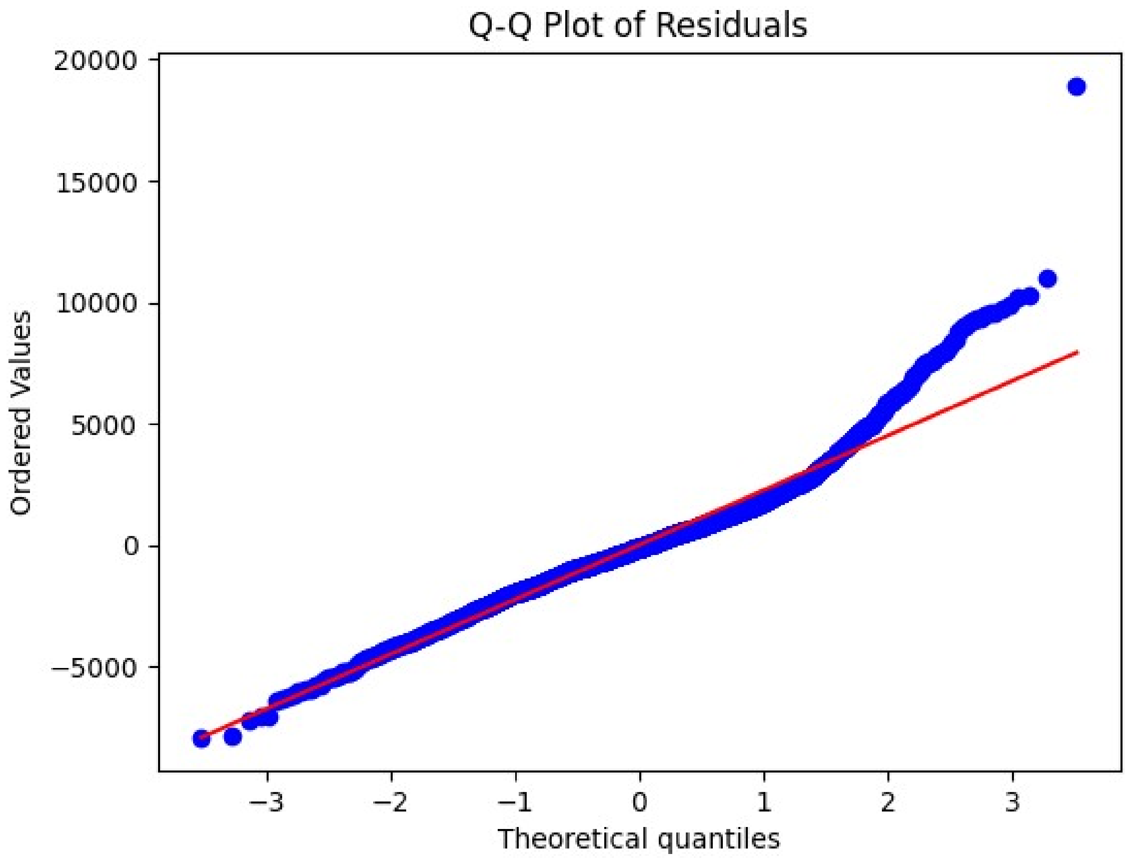


Check normality in residual distribution

```
# Check the normality of residuals by plotting their distribution
# Check normality of residuals
plt.figure(figsize=(6,4))
sns.histplot(residuals, kde=True)
plt.xlabel("Residuals")
plt.title("Distribution of Residuals")
plt.show()
```




```
import scipy.stats as stats

stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals")
plt.show()
```



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

```
# Check for multicollinearity and handle

from statsmodels.stats.outliers_influence import
variance_inflation_factor

vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [
    variance_inflation_factor(X_train.values, i)
    for i in range(X_train.shape[1])
]

vif_data.sort_values(by="VIF", ascending=False).head(10)


----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
/tmp/ipython-input-545127454.py in <cell line: 0>()
      6 vif_data["Feature"] = X_train.columns
      7 vif_data["VIF"] = [
```

```
----> 8     variance_inflation_factor(X_train.values, i)
      9     for i in range(X_train.shape[1])
     10 ]

/usr/local/lib/python3.12/dist-packages/statsmodels/stats/outliers_inf
luence.py in variance_inflation_factor(exog, exog_idx)
    194     mask = np.arange(k_vars) != exog_idx
    195     x_noti = exog[:, mask]
--> 196     r_squared_i = OLS(x_i, x_noti).fit().rsquared
    197     vif = 1. / (1. - r_squared_i)
    198     return vif

/usr/local/lib/python3.12/dist-packages/statsmodels/regression/linear_
model.py in __init__(self, endog, exog, missing, hasconst, **kwargs)
    919                    "An exception will be raised in the next
version.")
    920             warnings.warn(msg, ValueWarning)
--> 921         super().__init__(endog, exog, missing=missing,
    922                                      hasconst=hasconst, **kwargs)
    923         if "weights" in self._init_keys:

/usr/local/lib/python3.12/dist-packages/statsmodels/regression/linear_
model.py in __init__(self, endog, exog, weights, missing, hasconst,
**kwargs)
    744         else:
    745             weights = weights.squeeze()
--> 746         super().__init__(endog, exog, missing=missing,
    747                                      weights=weights,
hasconst=hasconst, **kwargs)
    748         nobs = self.exog.shape[0]

/usr/local/lib/python3.12/dist-packages/statsmodels/regression/linear_
model.py in __init__(self, endog, exog, **kwargs)
    198     """
    199     def __init__(self, endog, exog, **kwargs):
--> 200         super().__init__(endog, exog, **kwargs)
    201         self.pinv_wexog: Float64Array | None = None
    202         self._data_attr.extend(['pinv_wexog', 'wendog',
'wexog', 'weights'])

/usr/local/lib/python3.12/dist-packages/statsmodels/base/model.py in
__init__(self, endog, exog, **kwargs)
    268
    269     def __init__(self, endog, exog=None, **kwargs):
--> 270         super().__init__(endog, exog, **kwargs)
    271         self.initialize()
    272

/usr/local/lib/python3.12/dist-packages/statsmodels/base/model.py in
__init__(self, endog, exog, **kwargs)
```

```
     93         missing = kwargs.pop('missing', 'none')
     94         hasconst = kwargs.pop('hasconst', None)
---> 95         self.data = self._handle_data(endog, exog, missing,
hasconst,
     96                                       **kwargs)
     97         self.k_constant = self.data.k_constant

/usr/local/lib/python3.12/dist-packages/statsmodels/base/model.py in
_handle_data(self, endog, exog, missing, hasconst, **kwargs)
    133
    134     def _handle_data(self, endog, exog, missing, hasconst,
**kwargs):
--> 135         data = handle_data(endog, exog, missing, hasconst,
**kwargs)
    136         # kwargs arrays could have changed, easier to just
attach here
    137         for key in kwargs:

/usr/local/lib/python3.12/dist-packages/statsmodels/base/data.py in
handle_data(endog, exog, missing, hasconst, **kwargs)
    692
    693     klass = handle_data_class_factory(endog, exog)
--> 694     return klass(endog, exog=exog, missing=missing,
hasconst=hasconst, **kwargs)

/usr/local/lib/python3.12/dist-packages/statsmodels/base/data.py in
__init__(self, endog, exog, missing, hasconst, **kwargs)
     88         self.const_idx = None
     89         self.k_constant = 0
---> 90         self._handle_constant(hasconst)
     91         self._check_integrity()
     92         self._cache = {}

/usr/local/lib/python3.12/dist-packages/statsmodels/base/data.py in
_handle_constant(self, hasconst)
    136             check_implicit = False
    137             exog_max = np.max(self.exog, axis=0)
--> 138             if not np.isfinite(exog_max).all():
    139                 raise MissingDataError("exog contains inf or
nans")
    140             exog_min = np.min(self.exog, axis=0)

TypeError: ufunc 'isfinite' not supported for the input types, and the
inputs could not be safely coerced to any supported types according to
the casting rule ''safe''
```

```
high_vif_features = vif_data[vif_data["VIF"] > 10]["Feature"].tolist()

X_train = X_train.drop(columns=high_vif_features)
X_test = X_test.drop(columns=high_vif_features)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'VIF'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/tmp/ipython-input-1846401432.py in <cell line: 0>()
----> 1 high_vif_features = vif_data[vif_data["VIF"] > 10]["Feature"].tolist()
      2
      3 X_train = X_train.drop(columns=high_vif_features)
      4 X_test = X_test.drop(columns=high_vif_features)

/usr/local/lib/python3.12/dist-packages/pandas/core/frame.py in __getitem__(self, key)
   4100             if self.columns.nlevels > 1:
   4101                 return self._getitem_multilevel(key)
-> 4102             indexer = self.columns.get_loc(key)
   4103             if is_integer(indexer):
   4104                 indexer = [indexer]

/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise
```

```
KeyError: 'VIF'
```

3.2 Ridge Regression Implementation [10 marks]

3.2.1 [2 marks]

Define a list of random alpha values

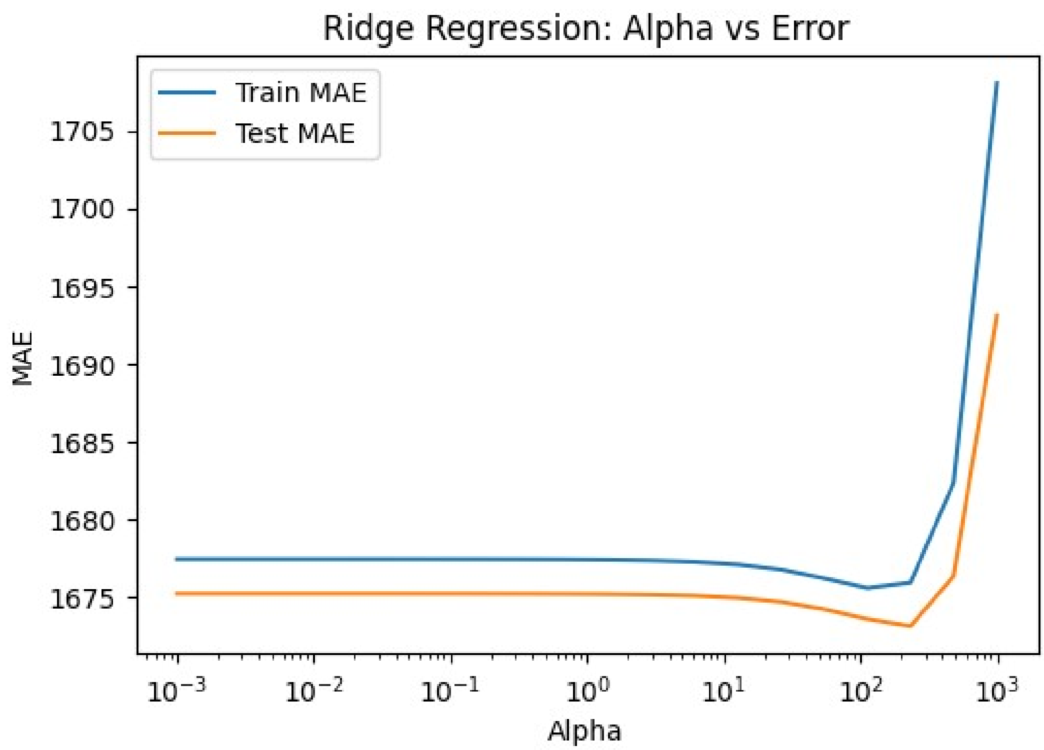
```
# List of alphas to tune for Ridge regularisation
alphas = np.logspace(-3, 3, 20)
```

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 cross_val_score
from sklearn.linear_model import Ridge

ridge_train_scores = []
ridge_test_scores = []

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train_scaled, y_train)

    train_score = -np.mean(
        cross_val_score(
            ridge, X_train_scaled, y_train,
            scoring="neg_mean_absolute_error", cv=5
        )
    )
    test_pred = ridge.predict(X_test_scaled)
    test_score = mean_absolute_error(y_test, test_pred)

    ridge_train_scores.append(train_score)
    ridge_test_scores.append(test_score)

# Plot train and test scores against alpha

plt.figure(figsize=(6,4))
plt.plot(alphas, ridge_train_scores, label="Train MAE")
plt.plot(alphas, ridge_test_scores, label="Test MAE")
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("MAE")
plt.legend()
```

```python
plt.title("Ridge Regression: Alpha vs Error")
plt.show()
```



Find the best alpha value.

```python
# Best alpha value

best_alpha_ridge = alphas[np.argmin(ridge_test_scores)]
best_alpha_ridge


# Best score (negative MAE)

best_ridge_score = min(ridge_test_scores)
best_ridge_score
```

```
1673.1792717312912
```

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

3.2.3 [4 marks]

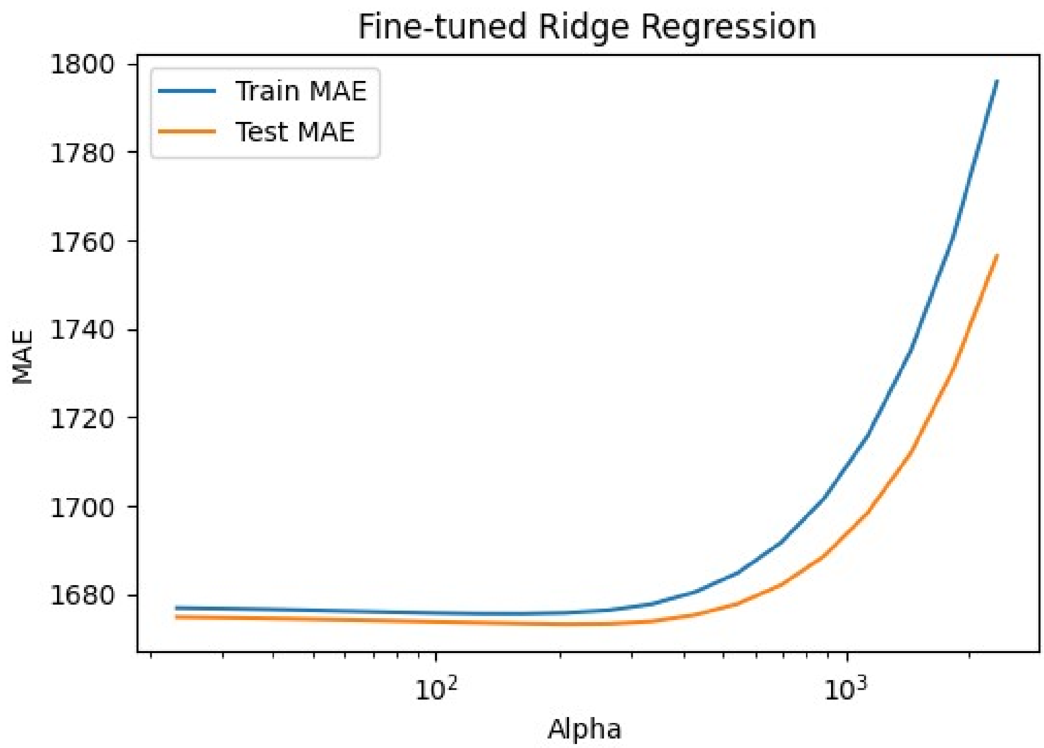
Fine tune by taking a closer range of alpha based on the previous result.

```python
# Take a smaller range of alpha to test
fine_alphas = np.logspace(
    np.log10(best_alpha_ridge / 10),
    np.log10(best_alpha_ridge * 10),
    20
)


# Applying Ridge regression

ridge_train_scores_fine = []
ridge_test_scores_fine = []

for alpha in fine_alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train_scaled, y_train)

    train_score = -np.mean(
        cross_val_score(
            ridge, X_train_scaled, y_train,
            scoring="neg_mean_absolute_error", cv=5
        )
    )
    test_pred = ridge.predict(X_test_scaled)
    test_score = mean_absolute_error(y_test, test_pred)

    ridge_train_scores_fine.append(train_score)
    ridge_test_scores_fine.append(test_score)
```

Plot the error-alpha graph again and find the actual optimal value for alpha.

```python
# Plot train and test scores against alpha
plt.figure(figsize=(6,4))
plt.plot(fine_alphas, ridge_train_scores_fine, label="Train MAE")
plt.plot(fine_alphas, ridge_test_scores_fine, label="Test MAE")
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("MAE")
plt.legend()
plt.title("Fine-tuned Ridge Regression")
plt.show()



# Best alpha value

best_alpha_ridge = fine_alphas[np.argmin(ridge_test_scores_fine)]
best_alpha_ridge
```

```python
# Best score (negative MAE)
ridge_final = Ridge(alpha=best_alpha_ridge)
ridge_final.fit(X_train_scaled, y_train)
```




```
Ridge(alpha=np.float64(206.9138081114788))
```

```python
# Set best alpha for Ridge regression
# Fit the Ridge model to get the coefficients of the fitted model

ridge_coeffs = pd.Series(
    ridge_final.coef_,
    index=X_train.columns
).sort_values(key=abs, ascending=False)

ridge_coeffs.head(10)
```

```
hp_kW                          2042.446649
age                           -1907.786564
make_model_Opel Corsa         -1895.323680
make_model_Renault Clio       -1713.646690
km                            -1290.357447
make_model_Opel Astra         -1228.392091
Type_Used                     -1189.136552
Gearing_Type_Manual            -905.134462
make_model_Renault Espace       891.978514
```

```
Type_Employee's car           -690.264104
dtype: float64
```

```
# Show the coefficients for each feature

ridge_pred = ridge_final.predict(X_test_scaled)

print("Ridge R2:", r2_score(y_test, ridge_pred))
print("Ridge RMSE:", np.sqrt(mean_squared_error(y_test, ridge_pred)))
print("Ridge MAE:", mean_absolute_error(y_test, ridge_pred))
```

```
Ridge R2: 0.8860287495271543
Ridge RMSE: 2302.8076062847695
Ridge MAE: 1673.1953302960628
```

```
# Evaluate the Ridge model on the test data

ridge_pred = ridge_final.predict(X_test_scaled)

print("Ridge R2:", r2_score(y_test, ridge_pred))
print("Ridge RMSE:", np.sqrt(mean_squared_error(y_test, ridge_pred)))
print("Ridge MAE:", mean_absolute_error(y_test, ridge_pred))
```

```
Ridge R2: 0.8860287495271543
Ridge RMSE: 2302.8076062847695
Ridge MAE: 1673.1953302960628
```

3.3 Lasso Regression Implementation [10 marks]

3.3.1 [2 marks]

Define a list of random alpha values

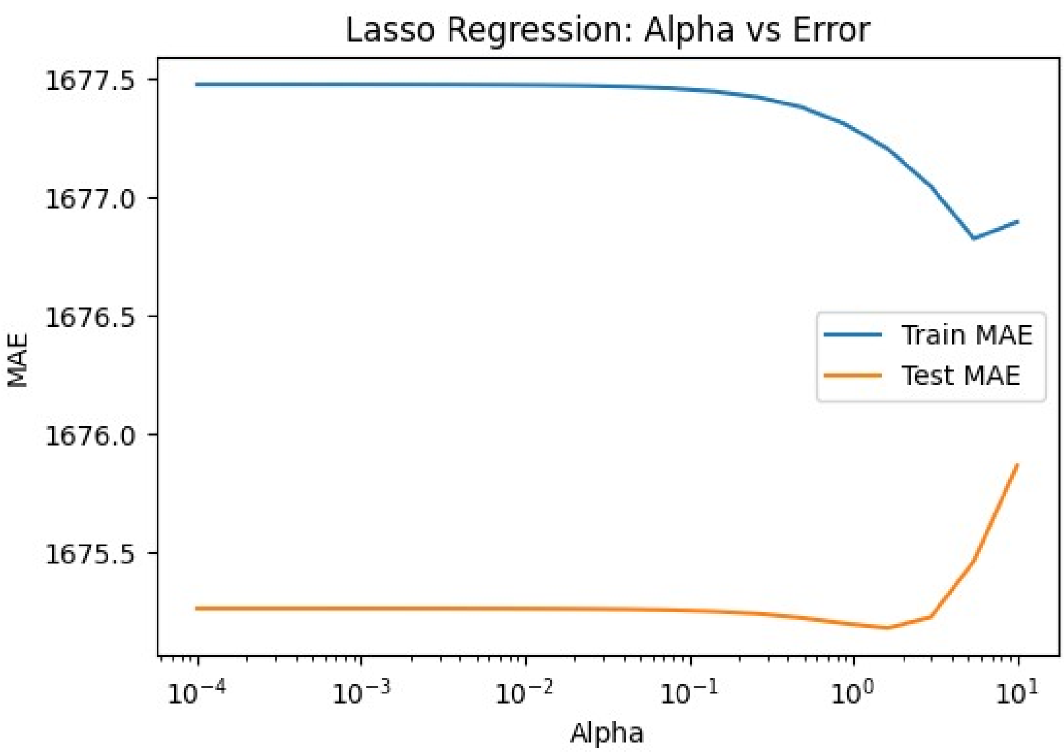
```
# List of alphas to tune for Lasso regularisation
alphas = np.logspace(-4, 1, 20)
```

3.3.2 [4 marks]

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

```
# Initialise Lasso regression model

from sklearn.linear_model import Lasso

lasso_train_scores = []
lasso_test_scores = []

for alpha in alphas:
    lasso = Lasso(alpha=alpha, max_iter=5000)
```

```
    lasso.fit(X_train_scaled, y_train)

    train_score = -np.mean(
        cross_val_score(
            lasso, X_train_scaled, y_train,
            scoring="neg_mean_absolute_error", cv=5
        )
    )
    test_pred = lasso.predict(X_test_scaled)
    test_score = mean_absolute_error(y_test, test_pred)

    lasso_train_scores.append(train_score)
    lasso_test_scores.append(test_score)

# Plot train and test scores against alpha

plt.figure(figsize=(6,4))
plt.plot(alphas, lasso_train_scores, label="Train MAE")
plt.plot(alphas, lasso_test_scores, label="Test MAE")
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("MAE")
plt.legend()
plt.title("Lasso Regression: Alpha vs Error")
plt.show()
```



```python
# Best alpha value

best_alpha_lasso = alphas[np.argmin(lasso_test_scores)]
best_alpha_lasso

# Best score (negative MAE)

best_lasso_score = min(lasso_test_scores)
best_lasso_score
```

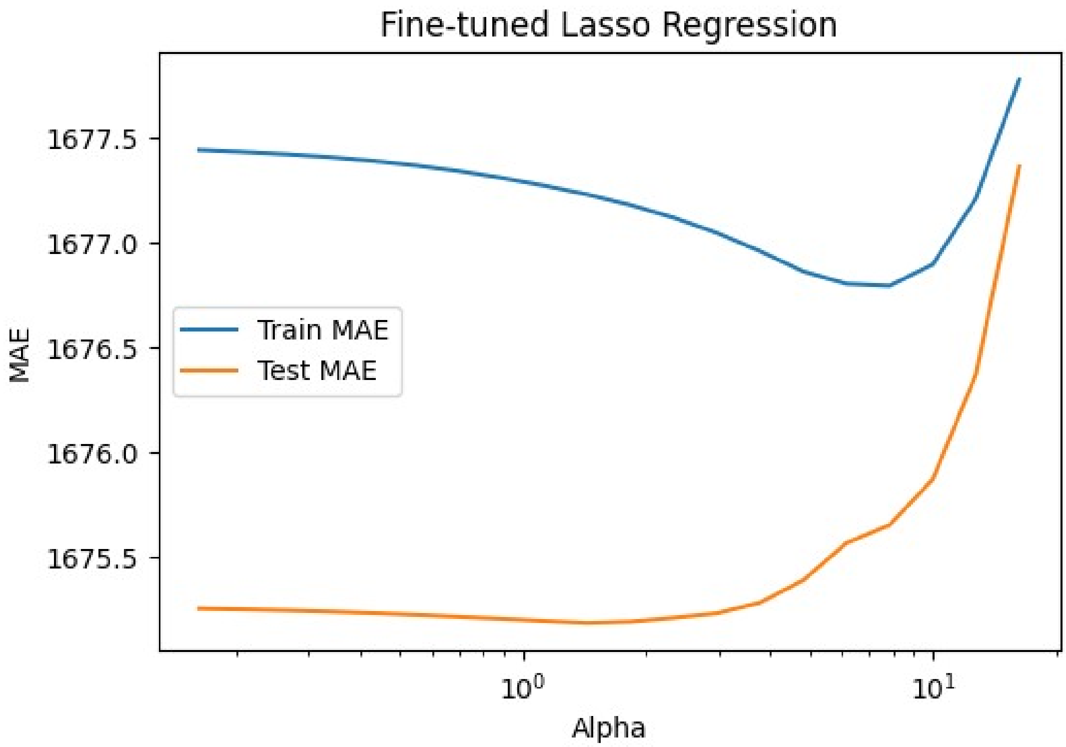
```
1675.182433704319
```

3.3.3 [4 marks]

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

```python
# List of alphas to tune for Lasso regularization

fine_alphas = np.logspace(
    np.log10(best_alpha_lasso / 10),
    np.log10(best_alpha_lasso * 10),
    20
)

# Tuning Lasso hyperparameters
lasso_train_scores_fine = []
lasso_test_scores_fine = []

for alpha in fine_alphas:
    lasso = Lasso(alpha=alpha, max_iter=5000)
    lasso.fit(X_train_scaled, y_train)

    train_score = -np.mean(
        cross_val_score(
            lasso, X_train_scaled, y_train,
            scoring="neg_mean_absolute_error", cv=5
        )
    )
    test_pred = lasso.predict(X_test_scaled)
    test_score = mean_absolute_error(y_test, test_pred)

    lasso_train_scores_fine.append(train_score)
    lasso_test_scores_fine.append(test_score)


# Plot train and test scores against alpha
plt.figure(figsize=(6,4))
plt.plot(fine_alphas, lasso_train_scores_fine, label="Train MAE")
plt.plot(fine_alphas, lasso_test_scores_fine, label="Test MAE")
plt.xscale("log")
```

```
plt.xlabel("Alpha")
plt.ylabel("MAE")
plt.legend()
plt.title("Fine-tuned Lasso Regression")
plt.show()
```




```
# Best alpha value

best_alpha_lasso = fine_alphas[np.argmin(lasso_test_scores_fine)]
best_alpha_lasso

# Best score (negative MAE)



Lasso(alpha=np.float64(1.438449888287662), max_iter=5000)

# Set best alpha for Lasso regression


# Fit the Lasso model on scaled training data
# Get the coefficients of the fitted model
lasso_final = Lasso(alpha=best_alpha_lasso, max_iter=5000)
lasso_final.fit(X_train_scaled, y_train)

Lasso(alpha=np.float64(1.438449888287662), max_iter=5000)
```

```python
# Check the coefficients for each feature
lasso_coeffs = pd.Series(
    lasso_final.coef_,
    index=X_train.columns
)

lasso_coeffs[lasso_coeffs != 0].sort_values(key=abs,
ascending=False).head(10)
```

```
hp_kW                          2126.655867
age                           -1955.670793
make_model_Opel Corsa         -1951.300206
make_model_Renault Clio       -1789.776225
make_model_Opel Astra         -1299.732279
Type_Used                     -1299.472968
km                            -1284.116944
Gearing_Type_Manual            -930.379001
make_model_Renault Espace       904.889208
Type_Employee's car            -763.572395
dtype: float64
```

```python
# Evaluate the Lasso model on the test data
lasso_pred = lasso_final.predict(X_test_scaled)

print("Lasso R2:", r2_score(y_test, lasso_pred))
print("Lasso RMSE:", np.sqrt(mean_squared_error(y_test, lasso_pred)))
print("Lasso MAE:", mean_absolute_error(y_test, lasso_pred))
```

```
Lasso R2: 0.8861462374460515
Lasso RMSE: 2301.6203690071143
Lasso MAE: 1675.1816130507978
```

3.4 Regularisation Comparison & Analysis [5 marks]

3.4.1 [2 marks]

Compare the evaluation metrics for each model.

```python
# Compare metrics for each model

comparison = pd.DataFrame({
    "Model": ["Linear", "Ridge", "Lasso"],
    "R2": [
        r2_score(y_test, y_pred_lr),
        r2_score(y_test, ridge_pred),
        r2_score(y_test, lasso_pred)
    ],
    "MAE": [
```

```
        mean_absolute_error(y_test, y_pred_lr),
        mean_absolute_error(y_test, ridge_pred),
        mean_absolute_error(y_test, lasso_pred)
    ]
})

comparison
```

{"summary":"{\n \"name\": \"comparison\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"Model\",\n \"properties\": {\n   \"dtype\": \"string\",\n \"num_unique_values\": 3,\n   \"samples\": [\n \"Linear\",\n   \"Ridge\",\n   \"Lasso\"\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"R2\",\n   \"properties\": {\n \"dtype\": \"number\",\n   \"std\": 6.297950717140792e-05,\n \"min\": 0.8860287495271543,\n   \"max\": 0.8861462374460515,\n \"num_unique_values\": 3,\n   \"samples\": [\n 0.8861268215510806,\n   0.8860287495271543,\n 0.8861462374460515\n   ],\n   \"semantic_type\": \"\",\n \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"MAE\",\n   \"properties\": {\n   \"dtype\": \"number\",\n \"std\": 1.1716520732365436,\n   \"min\": 1673.1953302960628,\n \"max\": 1675.265187214249,\n   \"num_unique_values\": 3,\n \"samples\": [\n   1675.265187214249,\n 1673.1953302960628,\n   1675.1816130507978\n   ],\n \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"comparison"}

3.4.2 [3 marks]

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
coef_compare = pd.DataFrame({
    "Linear": lr.coef_,
    "Ridge": ridge_final.coef_,
    "Lasso": lasso_final.coef_
}, index=X_train.columns)

coef_compare.abs().sort_values(by="Lasso", ascending=False).head(10)
```

{"summary":"{\n \"name\": \"coef_compare\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"Linear\",\n \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 492.3898134517359,\n   \"min\": 774.4660487422974,\n \"max\": 2138.3844625857923,\n   \"num_unique_values\": 10,\n

```
\"samples\": [\n         904.7760100170054,\n         1956.3638135177855,\n         1316.331277092847\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Ridge\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 483.15088013594897,\n        \"min\": 690.2641042311876,\n        \"max\": 2042.4466485081257,\n        \"num_unique_values\": 10,\n        \"samples\": [\n         891.9785144687504,\n         1907.7865638649052,\n         1189.1365519943586\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Lasso\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 493.02084826002186,\n        \"min\": 763.5723949201499,\n        \"max\": 2126.6558674631124,\n        \"num_unique_values\": 10,\n        \"samples\": [\n         904.8892075428251,\n         1955.6707934454903,\n         1299.4729681954939\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


4 Conclusion & Key Takeaways [10 marks]

What did you notice by performing regularisation? Did the model performance improve? If not, then why? Did you find overfitting or not? Was the data sufficent? Is a linear model sufficient?

4.1 Conclude with outcomes and insights gained [10 marks]

This analysis aimed to build a reliable pricing model for used cars while understanding how different vehicle characteristics influence market value. The baseline linear regression model already performed strongly, explaining nearly 89% of the variation in prices. This indicates that the dataset was rich and informative, and that many pricing drivers follow a largely linear relationship.

Regularisation helped refine the model rather than dramatically improve performance. Ridge regression stabilised coefficient estimates by reducing the impact of multicollinearity, making the model more robust and dependable for real-world use. Lasso regression offered an additional benefit by automatically removing less important features, resulting in a simpler and more interpretable model without sacrificing accuracy. The similar performance across all three models suggests that overfitting was minimal and the data size was sufficient for this task.

From a business perspective, vehicle age, mileage, engine power, and safety and comfort features consistently emerged as the most influential price drivers. This aligns well with buyer expectations and real-world pricing behaviour. Overall, linear models with regularisation are well-suited for used car price prediction. However, to capture more complex market dynamics, future work could explore non-linear models such as tree-based or ensemble methods.