
# Machine Learning Concepts
# (6G7V0015_2223_9F)

Student ID: 22547202
Name: Neha Hussain

# Data/Domain Understanding and Exploration

## Meaning and Type of Features; Analysis of Univariate Distributions

The dataset that we have been provided to work on is called **Car Sales Adverts** by AutoTrader. The dataset (adverts.csv) has 402,005 rows and 12 columns. The dataset has a total of 12 features. This dataset tells us about the characteristics of the cars that are up for sale and their prices.

The features that we can find in this dataset are **public_reference**, **mileage**, **reg_code**, **standard_colour**, **standard_make**, **standard_model**, **vehicle_condition**, **year_of_registration**, **price**, **body_type**, **crossover_car_and_van**, and **fuel_type.**

```
import pandas as pd
dataset = pd.read_csv('https://raw.githubusercontent.com/nehahussain/Ds_ML_dataset/main/adverts.csv')

print('Total number of columns: ' + str(dataset.shape[1]) + "\n" + "Total number of rows: " + str(dataset.shape[0]))
print(dataset.info())
```

```
Total number of columns: 12
Total number of rows: 402005
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402005 entries, 0 to 402004
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   public_reference      402005 non-null  int64
 1   mileage               401878 non-null  float64
 2   reg_code              370148 non-null  object
 3   standard_colour       396627 non-null  object
 4   standard_make         402005 non-null  object
 5   standard_model        402005 non-null  object
 6   vehicle_condition     402005 non-null  object
 7   year_of_registration  368694 non-null  float64
 8   price                 402005 non-null  int64
 9   body_type             401168 non-null  object
 10  crossover_car_and_van 402005 non-null  bool
 11  fuel_type             401404 non-null  object
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 34.1+ MB
None
```

The features in the dataset provide us with details about the vehicles that are up for sale. Here are some analyses regarding the features in the dataset. Firstly, we are describing the numerical features.

```
pd.set_option('display.float_format', '{:.2f}'.format)
dataset[['mileage', 'year_of_registration', 'price']].describe()
```

| | mileage | year_of_registration | price |
|---|---|---|---|
| **count** | 401878.00 | 368694.00 | 402005.00 |
| **mean** | 37743.60 | 2015.01 | 17341.97 |
| **std** | 34831.72 | 7.96 | 46437.46 |
| **min** | 0.00 | 999.00 | 120.00 |
| **25%** | 10481.00 | 2013.00 | 7495.00 |
| **50%** | 28629.50 | 2016.00 | 12600.00 |
| **75%** | 56875.75 | 2018.00 | 20000.00 |
| **max** | 999999.00 | 2020.00 | 9999999.00 |

The initial look at the values shows us that there are some erroneous values in the year_of_registration column. The minimum year in the dataset is 999 which is not possible. We will need to deep dive into this feature to fix it. Moreover, the max value of mileage and price need to be inspected as well.

This is what the categorical features look like:

```
categorical_features = ['reg_code', 'standard_colour', 'standard_make', 'standard_model',
             'vehicle_condition', 'body_type', 'crossover_car_and_van', 'fuel_type']
dataset[categorical_features].describe()
```

| | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | body_type | crossover_car_and_van | fuel_type |
|---|---|---|---|---|---|---|---|---|
| **count** | 370148 | 396627 | 402005 | 402005 | 402005 | 401168 | 402005 | 401404 |
| **unique** | 72 | 22 | 110 | 1168 | 2 | 16 | 2 | 9 |
| **top** | 17 | Black | BMW | Golf | USED | Hatchback | False | Petrol |
| **freq** | 36738 | 86287 | 37376 | 11583 | 370756 | 167315 | 400210 | 216929 |

Following screenshot tells the amount of missing values (nulls) in respective columns in the dataset.

```
dataset.isnull().sum()
```

```
public_reference          0
mileage                 127
reg_code              31857
standard_colour        5378
standard_make             0
standard_model            0
vehicle_condition         0
year_of_registration  33311
price                     0
body_type               837
crossover_car_and_van     0
fuel_type               601
dtype: int64
```

## Features in the dataset

1. **public_reference**

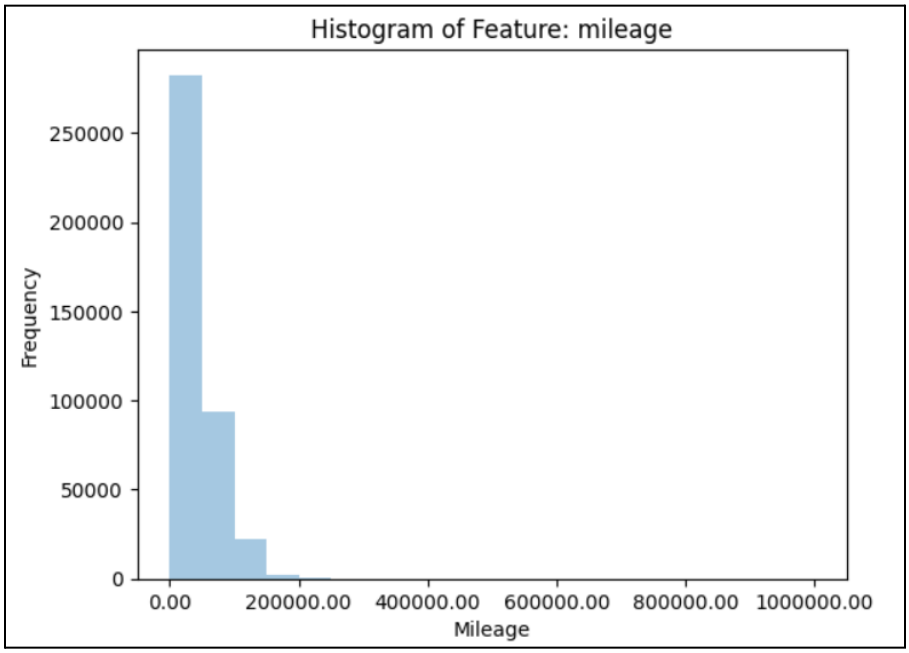
This is a 15-digit integer value in the dataset. Every record has a unique value. It can be assumed that this is a unique identifier that every advertisement has on AutoTrader. Hence, this may not play any part in predicting the price of the vehicle.

```
In [12]: # this code statement helps us to find if there are any duplicate values in the column
         pd.Series(dataset['public_reference']).is_unique
Out[12]: True
```

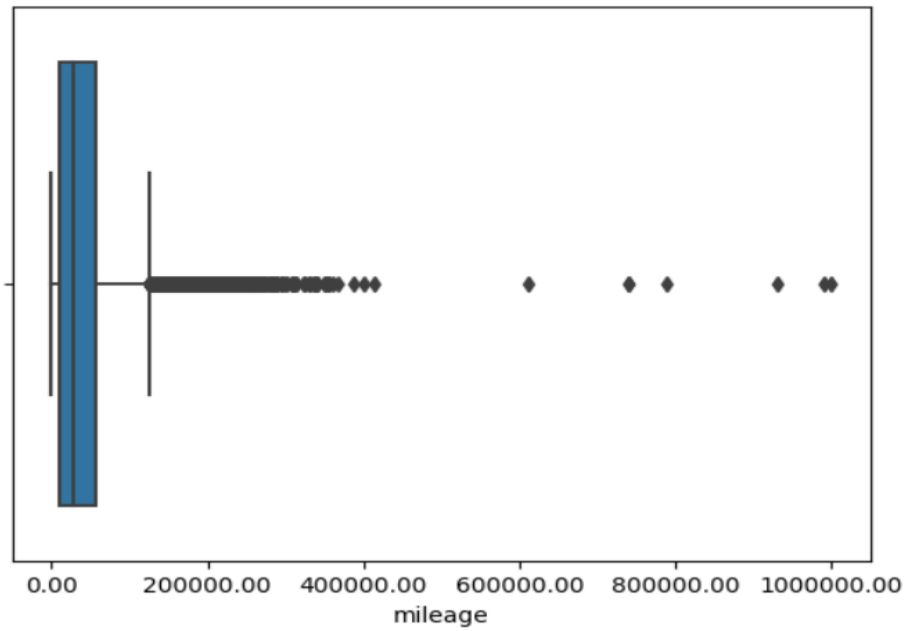
2. **mileage**

It is a numerical feature that tells about the miles clocked on the vehicle. This column has a float data type.

```
feature = 'mileage'
# Create a histogram of the data
sns.distplot(dataset[feature], kde=False, bins=20)

# Add labels and show plot
plt.xlabel('Mileage')
plt.ylabel('Frequency')
plt.title('Histogram of Feature: ' + feature)
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%.2f'))
plt.show()

# Calculate summary statistics
print("Summary statistics for feature: "+ feature)
print(dataset[feature].describe())

sns.boxplot(x=dataset[feature])
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%.2f'))
plt.show()
```




```
Summary statistics for feature: mileage
count   401,878.00
mean     37,743.60
std      34,831.72
min           0.00
25%      10,481.00
50%      28,629.50
75%      56,875.75
max     999,999.00
Name: mileage, dtype: float64
```



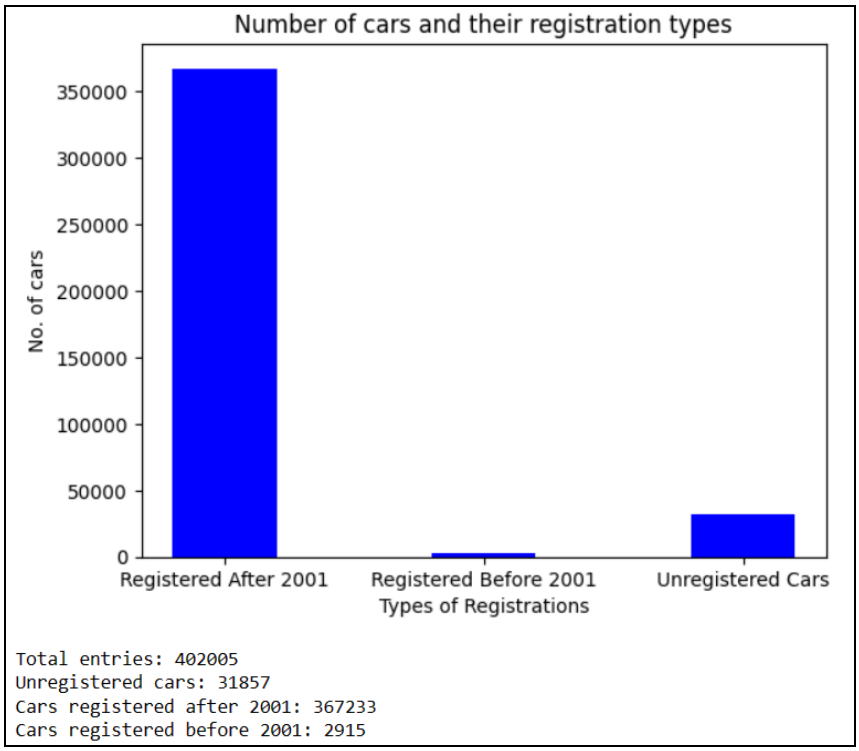

We can see outliers in the feature with the help of the boxplot above. Outliers and missing values in the feature will be handled later.

### 3. reg_code

This feature tells us about the age of the vehicle. Instead of the entire registration code of the vehicle, only the age identifier part of the code is available in this feature. There are two types of entries that can be found in this feature: **two-digit** integers and **single** alphabets. Vehicles registered from 1963 to 2001 had their age identifier in the alphabets. After 2001, the age identifier was changed to double digits.

The distribution of the data can be seen below:

```
import matplotlib.pyplot as plt

df_not_null = dataset[dataset['reg_code'].notnull() == True]
df_null = dataset[dataset['reg_code'].isna() == True]
df_alpha = df_not_null[df_not_null['reg_code'].str.isdigit() == False]
df_digits = df_not_null[df_not_null['reg_code'].str.isdigit() == True]

x_label = ['Registered After 2001', 'Registered Before 2001', 'Unregistered Cars']
y_label = [df_digits["reg_code"].count(), df_alpha["reg_code"].count(), df_null['reg_code'].isna().count()]
plt.bar(x_label, y_label, color ='blue',
        width = 0.4)
plt.xlabel("Types of Registrations")
plt.ylabel("No. of cars")
plt.title("Number of cars and their registration types")
plt.show()
print("Total entries: " + str(dataset.shape[0]))
print("Unregistered cars: " + str(df_null['reg_code'].isna().count()))
print("Cars registered after 2001: " + str(df_digits["reg_code"].count()))
print("Cars registered before 2001: " + str(df_alpha["reg_code"].count()))
```

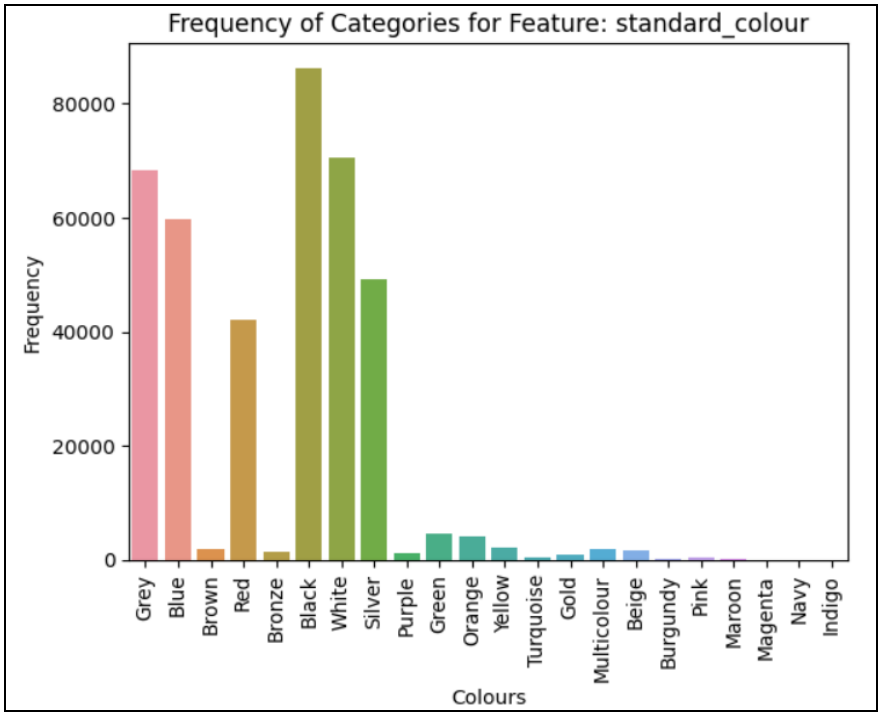



New cars that have not been registered yet have **null** values in the dataset.

4. **standard_colour**

It is a categorical feature that tells about the colour of the vehicle. The bar plot below offers an insight about the frequency of values in the feature.

```
# Create a bar plot of the frequency of each category
sns.countplot(x=dataset["standard_colour"])

# Add labels and show plot
plt.xlabel('Colours')
plt.ylabel('Frequency')
plt.title('Frequency of Categories for Feature: standard_colour')
plt.xticks(rotation=90)
plt.show()

print(f"Missing values: {dataset['standard_colour'].isnull().sum()}")
print(dataset['standard_colour'].describe())
```




```
Missing values: 5378
count     396627
unique        22
top        Black
freq       86287
Name: standard_colour, dtype: object
```

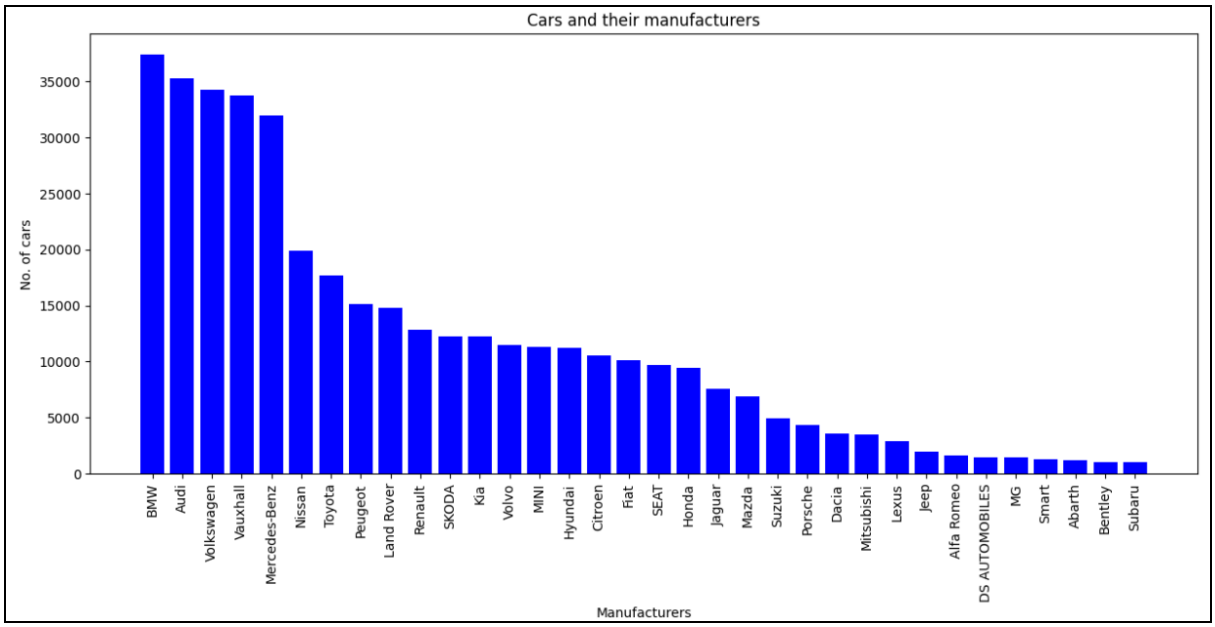
### 5. standard_make

This feature tells us about the manufacturer of the vehicle. It is a categorical variable. There are a total of 110 unique manufacturers in the dataset. German automakers BMW, Audi and Volkswagen have the most vehicles in this dataset.

Due to a large number of different manufacturers, only those manufacturers are included in the figure below that have more than 1000 vehicles in the entire dataset.

```
import matplotlib.pyplot as plt

vc = dataset['standard_make'].value_counts()
print("No. of manufacturers: " + str(len(vc)))
vc.index = vc.index.astype(str)
vc_dict = vc.to_dict()
x_label = []
y_label = []
for key in vc_dict.keys():
    if vc_dict[key] > 1000:
        x_label.append(key)
        y_label.append(vc_dict[key])

plt.figure(figsize=(15, 6))
plt.bar(x_label, y_label, color ='blue',
        width = 0.8)
ax = plt.gca()
ax.set_xticklabels(labels=x_label,rotation=90);
plt.xlabel("Manufacturers")
plt.ylabel("No. of cars")
plt.title("Cars and their manufacturers")
plt.show()
```

```
No. of manufacturers: 110
```




### 6. standard_model

This feature is closely related to the feature **standard_make**. It is a categorical feature. The model of the vehicle depends on the manufacturer. Volkswagen Golf has the most listings in the dataset.

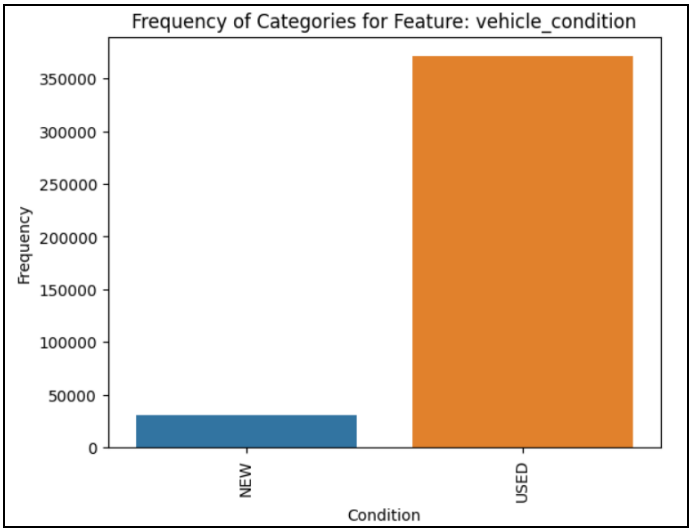
```
dataset['standard_model'].value_counts()
dataset.groupby(['standard_make','standard_model'])
group_size = dataset.groupby(['standard_make','standard_model']).size()
group_size.sort_values(ascending=False)
```

```
standard_make  standard_model
Volkswagen     Golf              11583
Vauxhall       Corsa             10646
Mercedes-Benz  C Class            8550
BMW            3 Series           8347
Volkswagen     Polo               7681
                                 ...
Toyota         Mark II Blitz         1
GMC            Pickup                1
Fiat           Uno                   1
Toyota         Paseo                 1
Porsche        917                   1
Length: 1217, dtype: int64
```

### 7. vehicle_condition

This is a categorical feature. It has only two possible values: **new** and **used**. This feature has an effect on the price of the vehicle. There are 370,756 used and 31,249 new vehicles.

```
# Create a bar plot of the frequency of each category
sns.countplot(x=dataset["vehicle_condition"])

# Add labels and show plot
plt.xlabel('Condition')
plt.ylabel('Frequency')
plt.title('Frequency of Categories for Feature: vehicle_condition')
plt.xticks(rotation=90)
plt.show()

print(dataset['vehicle_condition'].describe())
print(dataset['vehicle_condition'].value_counts())
```



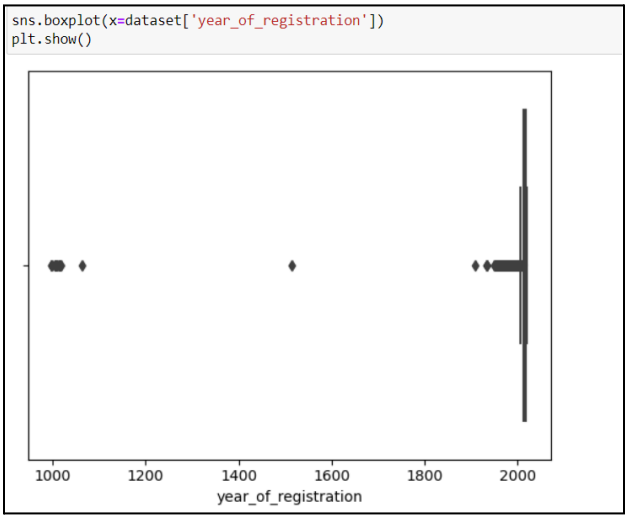

```
count     402005
unique         2
top         USED
freq      370756
Name: vehicle_condition, dtype: object
USED    370756
NEW      31249
Name: vehicle_condition, dtype: int64
```

**8. year_of_registration**

It is a categorical variable that tells us the year in which the vehicle was registered. It can be used to identify the age of the vehicle as well. New vehicles do not have a year of registration value. Hence, there are missing values for new vehicles. There are also some used vehicles that have null values as well.

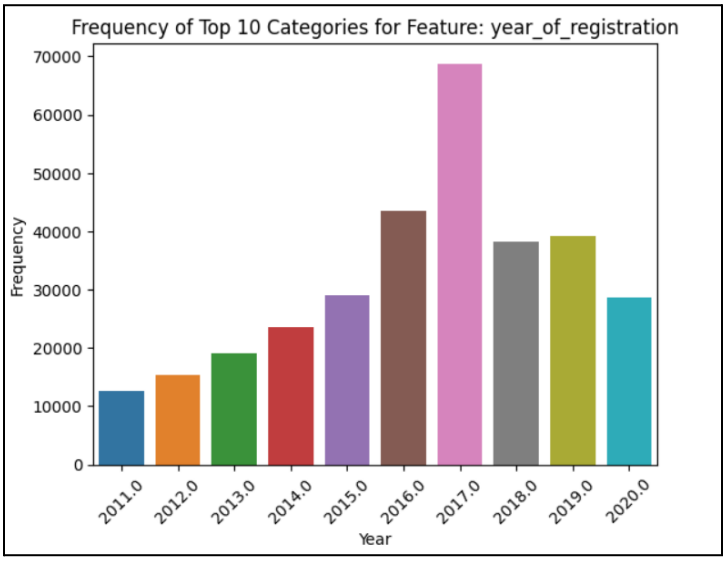
Moreover, there are erroneous values in this column. For example, there are records where the year of registration column has values like 1015, 1515 etc. We were able to find this out with the help of boxplot.

```
sns.boxplot(x=dataset['year_of_registration'])
plt.show()
```



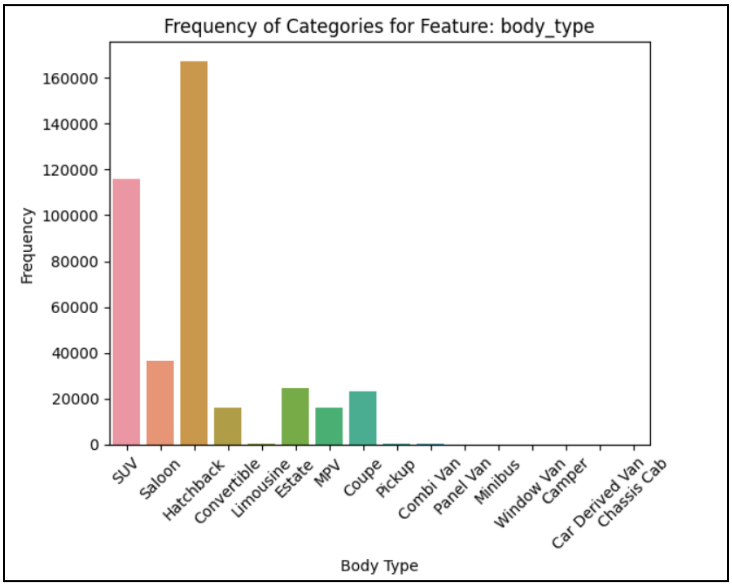
These values will be handled in the later stages of the report. Year 2017 has the most vehicles registered. As the data currently has noise, only top 10 values with the highest frequency are included in the bar plot.

```
feature = 'year_of_registration'
top_10 = dataset[feature].value_counts().head(10)

# Create a bar plot of the frequency of each category
sns.barplot(x=top_10.index, y=top_10.values)

# Add labels and show plot
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.title('Frequency of Top 10 Categories for Feature: ' + feature)
plt.show()
```



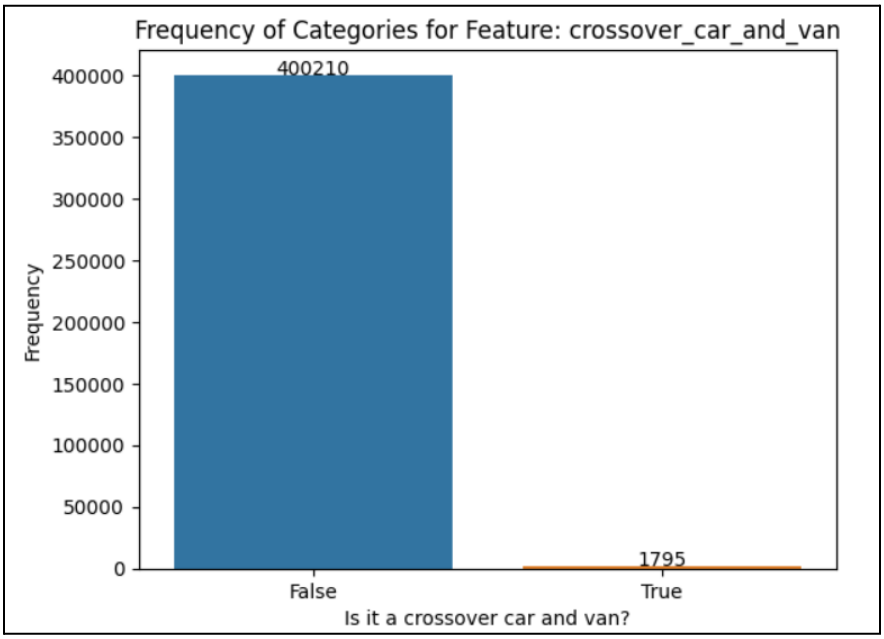

### 9. body_type

This is another categorical variable. It tells us about the body type of the vehicle. The body type gives us an idea about the size of the vehicle. For example, hatchbacks are smaller cars in comparison to SUV or saloons. We have some missing values in this column. These missing values can be attributed to the fact that they were not added when the advertisement was created.



**10. crossover_car_and_van**

This feature is a categorical variable. It populates only two types of values: True and False. There are no missing values in the column. This feature informs whether the vehicle up for sale is a crossover or a van. A crossover is a type of vehicle which is a hybrid of a hatchback and an SUV. It is smaller than an SUV but bigger than a hatchback.

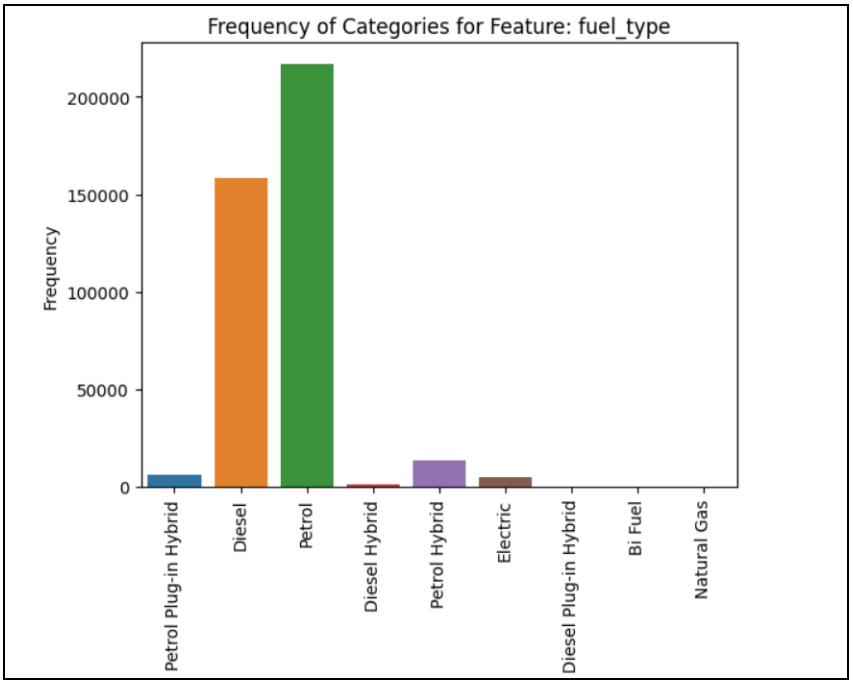



**11. fuel_type**

This feature is categorical and informs about the type of fuel that the vehicle uses to run. There are different fuel type options available in the dataset.

There are some missing values in the dataset for this particular feature as well. There is only one vehicle up for sale that uses natural gas as its fuel type. Most vehicles use either petrol or diesel.

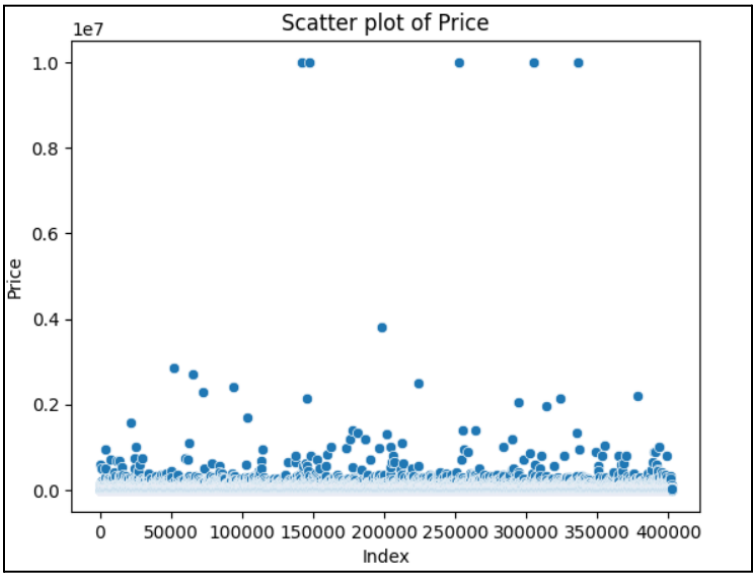
```
# Create a bar plot of the frequency of each body type
ax = sns.countplot(x=dataset["fuel_type"])

# Add labels and show plot
plt.xlabel('Fuel Type')
plt.ylabel('Frequency')
plt.title('Frequency of Categories for Feature: fuel_type')
plt.xticks(rotation=90)

plt.show()
```




**12. price**

Price is the target variable in the dataset. It is a continuous, numerical variable. There are no missing values in the column.

```
sns.scatterplot(data=dataset, x=dataset.index, y='price')
# sns.scatterplot(data=test_df, x=test_df['mileage'], y='price')

# Add labels and show plot
plt.xlabel('Index')
plt.ylabel('Price')
plt.title('Scatter plot of Price')
plt.show()

print(dataset['price'].describe())
```

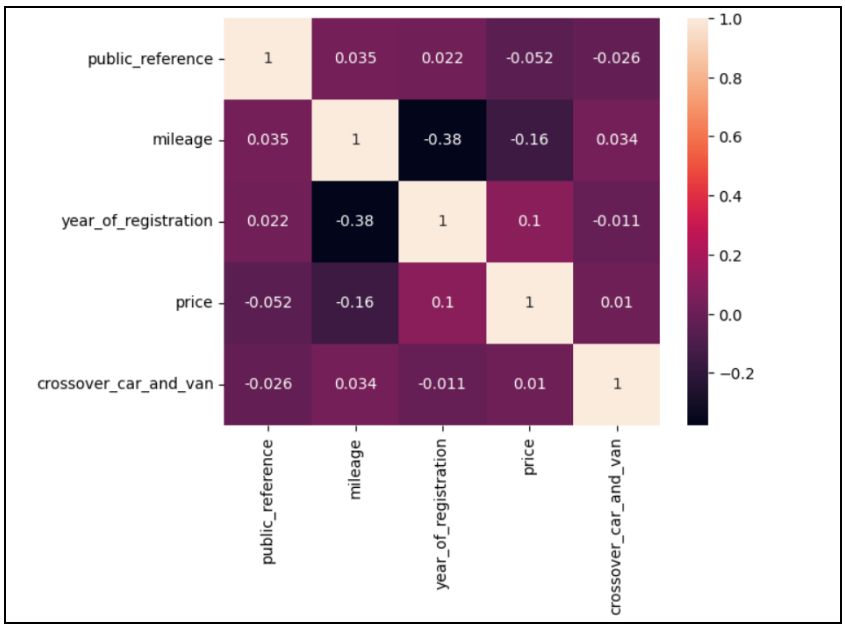


```
count      402,005.00
mean        17,341.97
std         46,437.46
min            120.00
25%          7,495.00
50%         12,600.00
75%         20,000.00
max      9,999,999.00
Name: price, dtype: float64
```

## Analysis of Predictive Power of Features

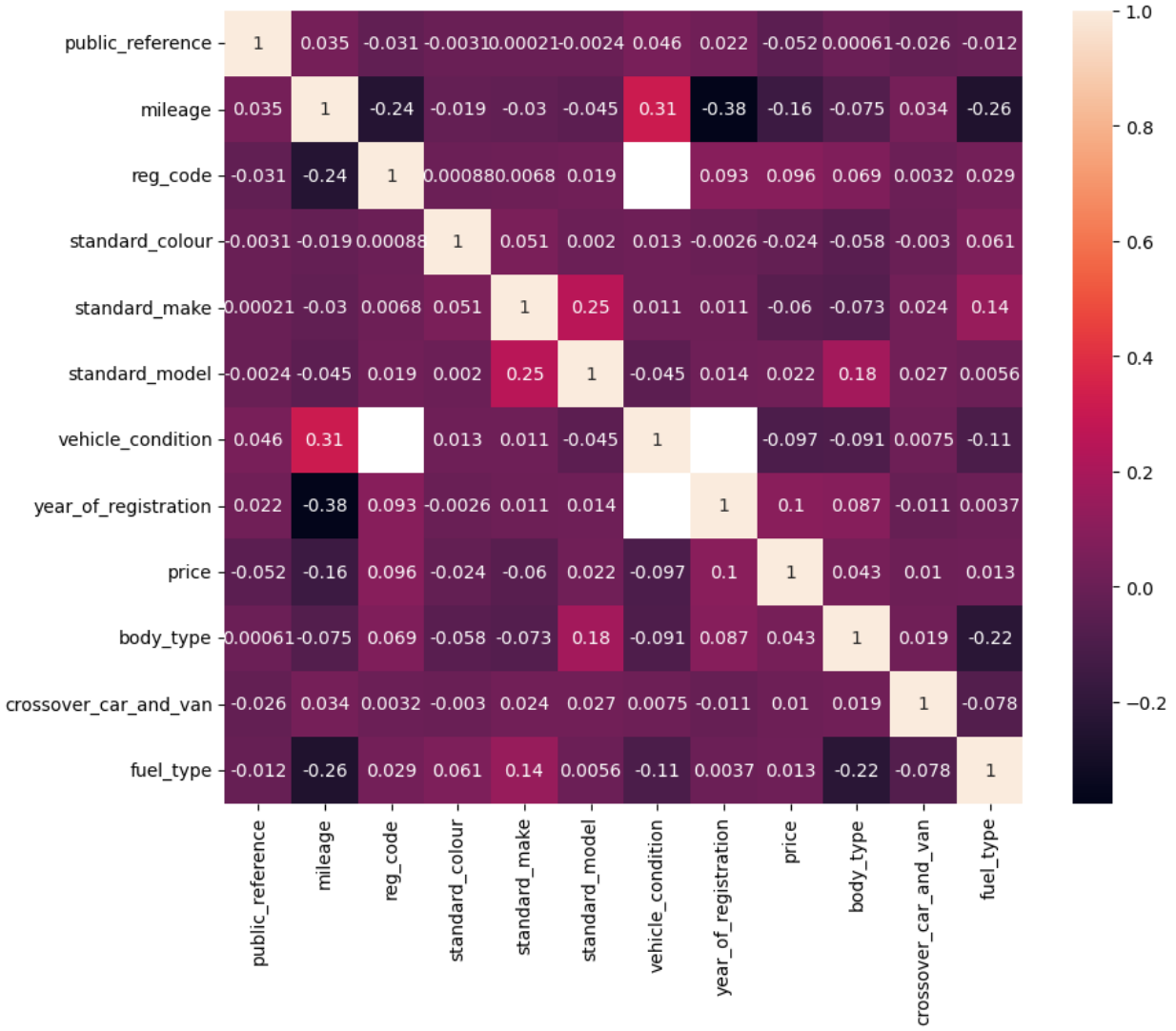
We will do feature importance to find the predictive power of the features. Correlation analysis will help to find relationships between the features and the target variable. The heatmap below provides correlation analysis results of numerical variables only. In this heatmap, you can see that mileage has a negative correlation with price. This makes sense as with increased mileage, a vehicle experiences wear and tear and depreciates in value.

```
corr = dataset.corr(numeric_only=True)

# Create a heatmap of the correlation matrix
sns.heatmap(corr, annot=True)

# Show the plot
plt.show()
```

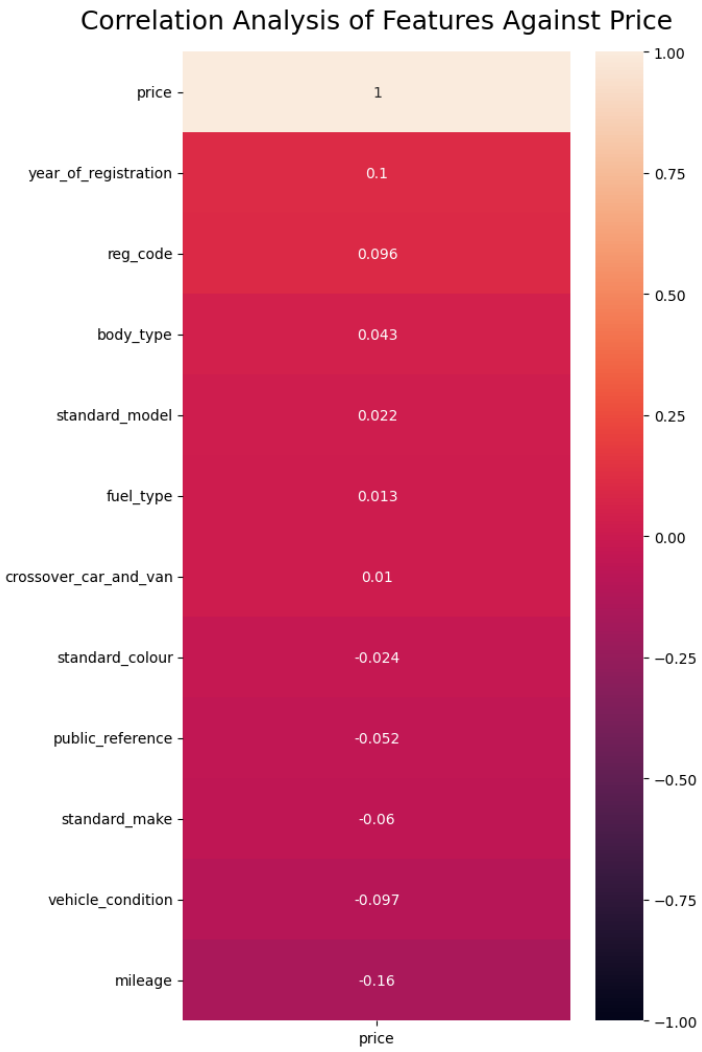



After running some data transformations and using ordinal encoding, we were able to do correlation analysis of all features. A function was made to handle the encoding of categorical features.

```
def encoder_function(feature_name: str, df):
    encoder = OrdinalEncoder()
    df[feature_name] = encoder.fit_transform(df[feature_name].values.reshape(-1,1))
```




A better outlook of correlation analysis of features against our target variable price can be seen below. We can conclude that mileage has the strongest correlation with price before the dataset is fixed by removing erroneous values and adding missing values.




We will run this correlation analysis again after fixing the dataset.

# Data Processing for Machine Learning

## Detection of Erroneous and Missing Values

Our initial analysis of the dataset helped us to find the following issues with the data:

- 9,999,999 in the price column for six records. It was also found that the year of registration and registration code values for those six records are also missing. Details regarding the erroneous values are mentioned in the ‘price’ subheading.

- Another erroneous value was found when looking for the maximum value in the ‘mileage’ column. The highest mileage in the dataset is recorded as 999,999. While clocking 999,999 miles/km on a vehicle is possible, the record raises some concerns because the asking price of the vehicle is 9,999 and the year of registration is 2013. An in-depth look at this particular record is done in the report earlier.

- Registration code column has null values. Null values are mostly for vehicles that are new. New vehicles for sale do not have a registration code assigned to them yet. However, we did find records where the vehicle condition was used but there was no registration code mentioned. There were null values instead. There are a total of **608** records of this type.

```
dataset[(dataset['reg_code'].isna() == True) & (dataset['vehicle_condition'] == 'USED')]
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **630** | 202010275479166 | 54000.0 | NaN | White | Toyota | Prius+ | USED | 2019.0 | 10900 | MPV |
| **682** | 202006019703585 | 103450.0 | NaN | Bronze | BMW | 3 Series | USED | 2006.0 | 6000 | Convertible |
| **1131** | 202009274274693 | 74000.0 | NaN | Silver | Mercedes-Benz | SL Class | USED | 2004.0 | 15950 | Convertible |
| **1194** | 202010114878094 | 13000.0 | NaN | White | Ferrari | 599 | USED | 2017.0 | 79949 | Coupe |
| **1335** | 202004018824812 | 76000.0 | NaN | White | Aston Martin | DB4 | USED | 2018.0 | 495000 | Saloon |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **397788** | 202009214000713 | 119400.0 | NaN | White | Toyota | Prius | USED | 2017.0 | 8995 | Hatchback |
| **397947** | 202010265437718 | 147898.0 | NaN | Blue | Toyota | Prado | USED | 2005.0 | 3250 | Estate |
| **398865** | 202010074728813 | 74500.0 | NaN | Silver | Toyota | Sienta | USED | 2020.0 | 4250 | MPV |
| **399728** | 202008262969804 | 12812.0 | NaN | Silver | Rolls-Royce | Wraith | USED | NaN | 159950 | Coupe |
| **400536** | 202010094805399 | 40523.0 | NaN | Red | Peugeot | 108 | USED | NaN | 5999 | Hatchback |

608 rows × 12 columns

- There are **5,378** records that have standard_colour missing.

```
dataset[dataset['standard_colour'].isna() == True].shape[0]
```

```
5378
```

- Year of registration is assigned to a vehicle as soon as it is brought onto the road. New vehicles for sale do not have a year of registration value in the dataset. They are missing in that case but due to logical reasons. However, there are **2,062** records that do not have a year of registration mentioned for them even though their condition is marked as ‘USED’ in the advertisement. To fix these missing values, we can use the registration code column to figure out the year of registration value. This will only be possible in cases where the registration code column is populated. There are **321** used vehicles in the dataset that do not have registration codes and years of registration added in their respective fields.

```
dataset[(dataset['year_of_registration'].isna() == True) & (dataset['vehicle_condition'] == 'USED')]
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **54** | 202007030806426 | 30000.0 | 18 | Red | Vauxhall | Insignia | USED | NaN | 11990 | Hatchback |
| **83** | 202008222801747 | 42847.0 | 61 | Red | Honda | Jazz | USED | NaN | 5695 | Hatchback |
| **865** | 202010084741550 | 43130.0 | 66 | White | Land Rover | Range Rover Sport | USED | NaN | 35990 | SUV |
| **968** | 202010305607535 | 63369.0 | 17 | Blue | SKODA | Rapid Spaceback | USED | NaN | 7490 | Hatchback |
| **1256** | 202010225294466 | 18715.0 | 68 | White | Volvo | V40 | USED | NaN | 16950 | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **400536** | 202010094805399 | 40523.0 | NaN | Red | Peugeot | 108 | USED | NaN | 5999 | Hatchback |
| **400725** | 202008252905656 | 45.0 | 20 | Silver | Land Rover | Range Rover Evoque | USED | NaN | 46945 | SUV |
| **401314** | 202010315635541 | 12522.0 | 67 | Red | Dacia | Sandero | USED | NaN | 6300 | Hatchback |
| **401323** | 201909222504136 | 46000.0 | 13 | Grey | Volkswagen | Caravelle | USED | NaN | 22995 | MPV |
| **401357** | 202007111114611 | 10.0 | 20 | Blue | Vauxhall | Corsa | USED | NaN | 16000 | Hatchback |

2062 rows × 12 columns

```
dataset[(dataset['year_of_registration'].isna() == True) & (dataset['vehicle_condition'] == 'USED') &
        (dataset['reg_code'].isna() == True)]
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **1510** | 202010054642656 | 13406.0 | NaN | White | Land Rover | Range Rover Evoque | USED | NaN | 26000 | Coupe |
| **2631** | 202010235350805 | 1000.0 | NaN | Blue | Maserati | Levante | USED | NaN | 63995 | SUV |
| **4766** | 202003238706011 | NaN | NaN | Grey | Subaru | Outback | USED | NaN | 35995 | Estate |
| **6998** | 202010225284269 | 160.0 | NaN | Grey | McLaren | Senna | USED | NaN | 699950 | Coupe |
| **7517** | 202009234093511 | 11413.0 | NaN | NaN | MINI | Convertible | USED | NaN | 14400 | Convertible |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **392499** | 202010064681927 | 83555.0 | NaN | Black | Land Rover | Range Rover Sport | USED | NaN | 30995 | SUV |
| **392730** | 202009093528195 | 38796.0 | NaN | Grey | Rover | 110 | USED | NaN | 3150 | Saloon |
| **396985** | 202001256559400 | 29000.0 | NaN | Black | Lamborghini | Gallardo | USED | NaN | 77990 | Coupe |
| **399728** | 202008262969804 | 12812.0 | NaN | Silver | Rolls-Royce | Wraith | USED | NaN | 159950 | Coupe |
| **400536** | 202010094805399 | 40523.0 | NaN | Red | Peugeot | 108 | USED | NaN | 5999 | Hatchback |

321 rows × 12 columns

- There are **837** records that have null values in the body type column.

```
dataset[dataset['body_type'].isna() == True]
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **307** | 202010245377951 | 33287.0 | 63 | Red | Volkswagen | Caddy | USED | 2013.0 | 10990 | NaN |
| **625** | 202010255419023 | 55000.0 | 65 | Grey | Vauxhall | Vivaro | USED | 2015.0 | 10995 | NaN |
| **1256** | 202010225294466 | 18715.0 | 68 | White | Volvo | V40 | USED | NaN | 16950 | NaN |
| **1643** | 202010305596351 | 24920.0 | 17 | Blue | BMW | 4 Series | USED | 2017.0 | 21980 | NaN |
| **1929** | 202010155047896 | 10.0 | 70 | Blue | Lotus | Elise | USED | 2020.0 | 47775 | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **399677** | 202007111130539 | 5001.0 | 20 | Black | Mercedes-Benz | GLC Class | USED | 2020.0 | 36870 | NaN |
| **400624** | 202010285545599 | 322000.0 | 05 | Black | London Taxis International | TXI | USED | 2005.0 | 995 | NaN |
| **400643** | 202011015665976 | 10.0 | NaN | White | Audi | A3 | NEW | NaN | 27845 | NaN |
| **400724** | 202008242879192 | 0.0 | L | Silver | Porsche | 911 | USED | 1973.0 | 175000 | NaN |
| **401788** | 202009103544825 | 67558.0 | 16 | Red | Mercedes-Benz | 220 | USED | 2016.0 | 12000 | NaN |

837 rows × 12 columns

- There are **601** records that have null values in their fuel_type column.

```
print(f"{dataset[dataset['fuel_type'].isna() == True].shape[0]} records have null values in fuel_type column")
dataset[dataset['fuel_type'].isna() == True]
```

```
601 records have null values in fuel_type column
```

| ileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type | crossover_car_and_van | fuel_type |
|---|---|---|---|---|---|---|---|---|---|---|
| 10.0 | NaN | Black | BMW | 5 Series | NEW | NaN | 51395 | Estate | False | NaN |
| 0.0 | NaN | NaN | Jaguar | XF | NEW | NaN | 35990 | Estate | False | NaN |
| 10.0 | NaN | Grey | Volvo | V60 | NEW | NaN | 31414 | Estate | False | NaN |
| 1568.0 | 14 | Silver | Toyota | Prius | USED | 2020.0 | 7995 | Hatchback | False | NaN |
| 10.0 | NaN | Black | Mercedes-Benz | A Class | NEW | NaN | 36584 | Hatchback | False | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0.0 | NaN | NaN | Jaguar | XF | NEW | NaN | 32585 | Saloon | False | NaN |
| 1.0 | NaN | Black | Peugeot | 508 | NEW | NaN | 39135 | Hatchback | False | NaN |
| 17.0 | 19 | NaN | McLaren | Senna | USED | 2019.0 | 799900 | NaN | False | NaN |
| 10.0 | NaN | Black | Vauxhall | Grandland X | NEW | NaN | 31190 | SUV | False | NaN |
| 1000.0 | 70 | Silver | Mercedes-Benz | A Class | USED | 2020.0 | 32000 | Hatchback | False | NaN |

- The year of registration column has some values that date back as far as 1007. This is an obvious error in data entry. On inspecting such records, it was easy to find the error. The registration code column provided the correct age and year of registration of the car. The first record in the example below has 1007 in the **year_of_registration** column. We can use the reg_code to find out the correct year of registration. The 07 means that the correct year of registration is **2007**. The same method can be used to find the correct year of registration for other entries as well.

```
vc = dataset['year_of_registration'].value_counts()
dataset[dataset['year_of_registration'] < 1900]
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **59010** | 202006270588110 | 14000.0 | 07 | Blue | Toyota | Prius | USED | 1007.0 | 7000 | Hatchback |
| **69516** | 202010155035879 | 96659.0 | 65 | Black | Audi | A4 Avant | USED | 1515.0 | 10385 | Estate |
| **84501** | 202009163810376 | 37771.0 | 63 | Black | Smart | fortwo | USED | 1063.0 | 4785 | Coupe |
| **114737** | 202008102305925 | 30000.0 | 59 | Red | Toyota | AYGO | USED | 1009.0 | 4695 | Hatchback |
| **120858** | 202010064654489 | 27200.0 | 66 | Black | MINI | Clubman | USED | 1016.0 | 18990 | Estate |
| **190556** | 202010205206488 | 58470.0 | 10 | Black | Fiat | Punto Evo | USED | 1010.0 | 3785 | Hatchback |
| **199830** | 202009013167637 | 23000.0 | 59 | Silver | MINI | Hatch | USED | 1009.0 | 5995 | Hatchback |

## Handling Outliers, Erroneous and Missing Values

To handle missing values in our dataset, we will be using mean, median and mode to fill them. The decision to use either mean, median or mode will depend on the type of data and skewness. Categorical variables can be filled using the mode value of the column. If the feature is numeric and the data is not skewed, the mean can be used to fill the null values. Otherwise, we are going to use the median.
We ran some analysis to check for skewness in our **mileage** and **year_of_registration** column.

```
print("Records with null mileage value: " + str(dataset[dataset["mileage"].isna() == True].shape[0]))
print("Skewness in mileage column: " +
      str(dataset['mileage'].skew(axis=0, skipna=True).round(decimals=2)))
# As the mileage column is skewed, we will need to use median to fill the NaN (null) values
print(dataset['mileage'].median())   # mileage median
```

```
Records with null mileage value: 127
Skewness in mileage column: 1.45
28629.5
```

```
print("Records with null year_of_registration value: "
      + str(dataset[(dataset["year_of_registration"].isna() == True)
                    & (dataset["vehicle_condition"] == "USED")].shape[0]))
print("Skewness in year_of_registration column: " +
      str(dataset['year_of_registration'].skew(axis=0, skipna=True).round(decimals=2)))
# As the year_of_registration column is skewed, we will need to use median to fill the NaN (null) values
print(dataset['year_of_registration'].median())
```

```
Records with null year_of_registration value: 2062
Skewness in year_of_registration column: -87.91
2016.0
```

Moreover, there were about 17 records that had incorrect year_of_registration values (the year before the 1900s).

```
dataset[dataset["year_of_registration"]<1900]["year_of_registration"].value_counts().sum()
```

```
17
```

To fix these erroneous values, we use the reg_code column. The reg_code column has an age identifier that allows us to figure out the correct year_of_registration value. To fill the missing values in the column, a mapping was created using reg_code and information from this website [1].

The erroneous values were fixed first. Then, the missing values were filled in where the vehicle condition is used. New vehicles do not have a year of registration. Then, records that had missing reg_code and year_of_registration, in those cases, year_of_registration was filled with 2016 (median of year_of_registration).

```
dataset[ (dataset['reg_code'].isna() == False) & (dataset['year_of_registration'].isna() == True) ]
```

| mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type | crossover_car_and_van | fuel_typ |
|---|---|---|---|---|---|---|---|---|---|---|
| 30000.0 | 18 | Red | Vauxhall | Insignia | USED | NaN | 11990 | Hatchback | False | Petr |
| 42847.0 | 61 | Red | Honda | Jazz | USED | NaN | 5695 | Hatchback | False | Petr |
| 43130.0 | 66 | White | Land Rover | Range Rover Sport | USED | NaN | 35990 | SUV | False | Dies |
| 63369.0 | 17 | Blue | SKODA | Rapid Spaceback | USED | NaN | 7490 | Hatchback | False | Dies |
| 18715.0 | 68 | White | Volvo | V40 | USED | NaN | 16950 | NaN | False | Petr |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 66287.0 | 63 | Blue | Vauxhall | Astra GTC | USED | NaN | 8400 | Coupe | False | Petr |
| 45.0 | 20 | Silver | Land Rover | Range Rover Evoque | USED | NaN | 46945 | SUV | False | Dies |
| 12522.0 | 67 | Red | Dacia | Sandero | USED | NaN | 6300 | Hatchback | False | Petr |
| 46000.0 | 13 | Grey | Volkswagen | Caravelle | USED | NaN | 22995 | MPV | False | Dies |
| 10.0 | 20 | Blue | Vauxhall | Corsa | USED | NaN | 16000 | Hatchback | False | Petr |

```
# Fixing year_of_registration column
reg_code_mapping = {
    '02': 2002,'03': 2003,'04': 2004,'05': 2005,'06': 2006,'07': 2007,'08': 2008,'09': 2009,
    '10': 2010,'11': 2011,'12': 2012,'13': 2013,'14': 2014,'15': 2015,'16': 2016,'17': 2017,'18': 2018,'19': 2019,'20': 2020,
    '51': 2001,'52': 2002, '53': 2003,'54': 2004,'55': 2005,'56': 2006,'57': 2007,'58': 2008,'59': 2009,
    '60': 2010,'61': 2011,'62': 2012,'63': 2013,'64': 2014,'65': 2015,'66': 2016,'67': 2017,'68': 2018,'69': 2019,'70': 2020
}

for index, row in dataset.iterrows():
    if row['reg_code'] in reg_code_mapping:
        dataset.at[index, 'year_of_registration'] = reg_code_mapping[row['reg_code']]
```

```
for index, row in dataset.iterrows():
    if pd.isnull(row['year_of_registration']) and row['vehicle_condition'] == 'USED':
        dataset.at[index, 'year_of_registration'] = 2016

# dataset[(dataset1['year_of_registration'].isna() == True) & (dataset['vehicle_condition'].isna() == 'USED')]
# the above statement will give empty dataframe to show no missing year_of_registration values for used vehicles
```

```
dataset[ (dataset['reg_code'].isna() == False) & (dataset['year_of_registration'].isna() == True)]  #no result, hence its filled
```

| public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type | crossover_ca |
|---|---|---|---|---|---|---|---|---|---|---|

We have handled the year of registration by reg_code, but we can not do the vice versa as we also need the months with a year of registration to use it to fill reg_code since the reg_code age identifier, changes twice a year, on the 1st of March and September [1].

The rest of the columns are categorical variables, hence, we found out their respective modes.

```
print("Mode of standard_colour: " + str(dataset['standard_colour'].mode()[0]))
print("\nMode of body_type: " + str(dataset['body_type'].mode()[0]))
print("\nMode of fuel_type: " + str(dataset['fuel_type'].mode()[0]))
```

```
Mode of standard_colour: Black

Mode of body_type: Hatchback

Mode of fuel_type: Petrol
```

The null values were then replaced with their respective values.

```
dataset = dataset.fillna({'mileage': dataset['mileage'].median(), 'standard_colour': 'Black',
                'body_type': 'Hatchback', 'fuel_type': 'Petrol'})
```

Now to handle the missing values where mileage is zero, means that the car has not been registered yet, we will replace the nan values with zero.

```
# Update the reg_code column where mileage is 0
dataset['reg_code'] = np.where(dataset.mileage == 0, 0, dataset['reg_code'])

# Update the year_of_registration column where mileage is 0
dataset['year_of_registration'] = np.where(dataset.mileage == 0, 0, dataset['year_of_registration'])
```

Next, we handle outliers in mileage and price columns of our dataset using the interquartile range method. This is crucial for the robustness of the model. We will filter out records that are outliers.

```
# Calculate Interquartile Range for mileage and price to deal with outliers
mileage_Q1 = dataset['mileage'].quantile(0.25)
mileage_Q3 = dataset['mileage'].quantile(0.75)
mileage_IQR = mileage_Q3 - mileage_Q1

price_Q1 = dataset['price'].quantile(0.25)
price_Q3 = dataset['price'].quantile(0.75)
price_IQR = price_Q3 - price_Q1

# Find the lower and upper bounds for outliers
mileage_lower_bound = mileage_Q1 - 1.5 * mileage_IQR
mileage_upper_bound = mileage_Q3 + 1.5 * mileage_IQR

price_lower_bound = price_Q1 - 1.5 * price_IQR
price_upper_bound = price_Q3 + 1.5 * price_IQR

# Filter out the outliers in mileage and price
dataset = dataset[(dataset['mileage'] > mileage_lower_bound) /
                  & (dataset['mileage'] < mileage_upper_bound) & (dataset['price'] > price_lower_bound) /
                  & (dataset['price'] < price_upper_bound)]
```

After filtering out the dataset, we will move on to ordinal encoding our dataset so that it is ready for the machine learning algorithms. We will be using the encoder_function that we have created (mentioned earlier in the report).

**Ordinal Encoding Categorical Columns**

```
encoder_function('reg_code', dataset)
encoder_function('standard_colour', dataset)
encoder_function('standard_make', dataset)
encoder_function('standard_model', dataset)
encoder_function('vehicle_condition', dataset)
encoder_function('body_type', dataset)
encoder_function('vehicle_condition', dataset)
encoder_function('fuel_type', dataset)
```

## Normalizing mileage and price

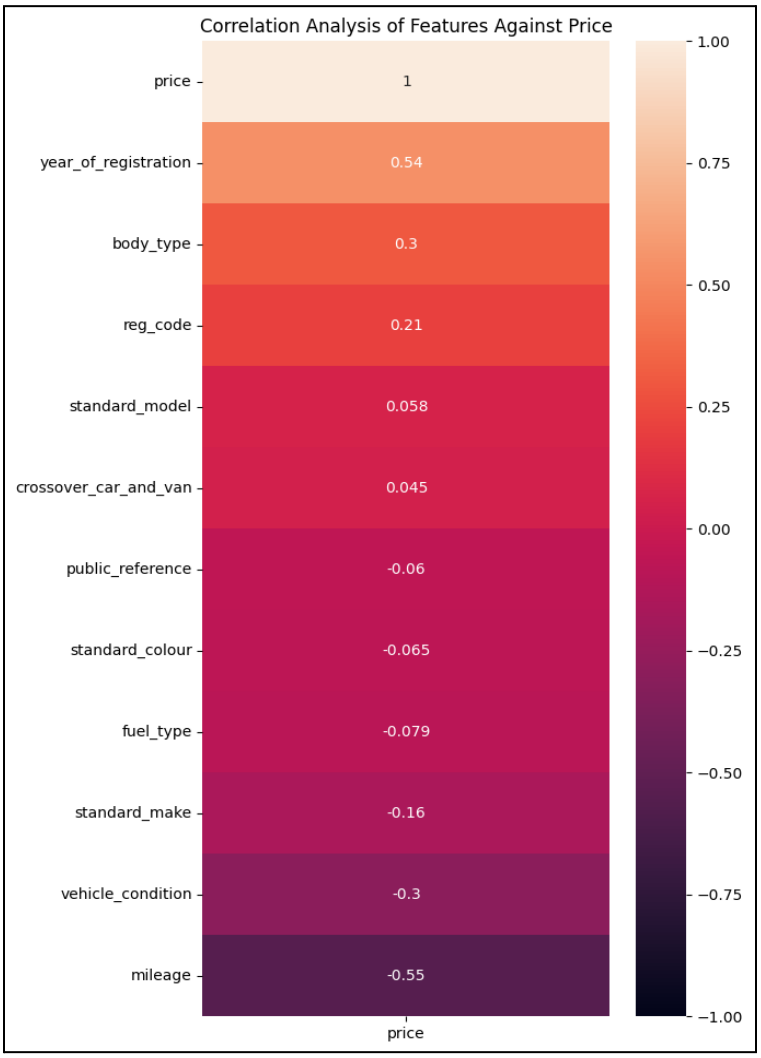
We are using MinMax to normalize values in mileage and price columns. The idea is to normalize the values between 0 to 1.

```
dataset_normalized=dataset.copy(deep=True)
dataset_normalized['mileage'] = (dataset_normalized['mileage'] - dataset_normalized['mileage'].min()) / (dataset_normalized['mile
dataset_normalized['price'] = (dataset_normalized['price'] - dataset_normalized['price'].min()) / (dataset_normalized['price'].ma
```

```
dataset_normalized[['mileage', 'price']].describe()
```

| | mileage | price |
|---|---|---|
| count | 367557.00 | 367557.00 |
| mean | 0.29 | 0.35 |
| std | 0.24 | 0.21 |
| min | 0.00 | 0.00 |
| 25% | 0.10 | 0.19 |
| 50% | 0.24 | 0.31 |
| 75% | 0.45 | 0.48 |
| max | 1.00 | 1.00 |

Now, before we proceed further with model building, let's see if our data processing has led to any changes in the correlation of features. Running the correlation analysis again showed us that the relationships got stronger after data processing.




# Feature Engineering

To help with predicting trends and patterns in our data, there can be new features engineered from existing features. Some new features were added to the dataset to provide better insights when looking at trends and averages of price when compared to different features.

### mileage_type

This feature is created to provide low, medium, or high values depending on the mileage of the vehicle. Vehicles with low mileage are more likely to sell for a higher value. To check whether this trend exists in our dataset, this new feature was created. For encoding purposes, we will be using 0 for low, 1 for medium and 2 for high.

Some analysis was required before a certain threshold could be set to decide whether the vehicle has low, medium, or high mileage. The feature engineering was done after the initial bit of data processing to add missing values and remove erroneous records.
The describe function on the mileage feature helps to provide a deep dive on the dispersion of the data.

```
dataset_normalized['mileage'].describe()
```

```
count   367557.00
mean         0.29
std          0.24
min          0.00
25%          0.10
50%          0.24
75%          0.45
max          1.00
Name: mileage, dtype: float64
```

According to the statistics above, the average mileage of the vehicles is about 0.29. As the values are normalized, we have decided to use the following values to determine whether the type of mileage is low, medium or high.

```
def categorize_mileage(mileage):
    if mileage < 0.3:
        return 0
    elif mileage >= 0.3 and mileage < 0.6:
        return 1
    else:
        return 2

dataset_normalized['mileage_type'] = dataset_normalized['mileage'].apply(categorize_mileage)
```

```
dataset_normalized[['standard_make','mileage','mileage_type', 'price']]
```

| | standard_make | mileage | mileage_type | price |
|---|---|---|---|---|
| **1** | 38.00 | 0.86 | 2 | 0.18 |
| **2** | 77.00 | 0.06 | 0 | 0.36 |
| **3** | 89.00 | 0.36 | 1 | 0.20 |
| **4** | 44.00 | 0.51 | 1 | 0.70 |
| **5** | 5.00 | 0.13 | 0 | 0.75 |

### vehicle_age

Age of the vehicle plays a part in the price of the vehicle. A new car will fetch a greater price than a car that is older. Except for some cases where the vehicle is rare, the price of the vehicle depreciates with age. This new feature can be a good indicator of the price of a vehicle. It is created using the year_of_registration column. To calculate the age of the vehicle, 2022 is used as the base year value. So, for example, if the year_of_registration is 2020, the vehicle age will be 2 years. New vehicles will have 0 in the column.

```
def vehicle_age_calculation(year_of_registration):
    if pd.isnull(year_of_registration):
        return 0
    else:
        return (2022-year_of_registration)

dataset_normalized['vehicle_age'] = dataset_normalized['year_of_registration'].apply(vehicle_age_calculation)
```

```
dataset_normalized[['mileage', 'price', 'year_of_registration', 'vehicle_age']]
```

| | mileage | price | year_of_registration | vehicle_age |
|---|---|---|---|---|
| **1** | 0.86 | 0.18 | 2011.00 | 11.00 |
| **2** | 0.06 | 0.36 | 2017.00 | 5.00 |
| **3** | 0.36 | 0.20 | 2016.00 | 6.00 |
| **4** | 0.51 | 0.70 | 2014.00 | 8.00 |
| **5** | 0.13 | 0.75 | 2017.00 | 5.00 |

# Model Building, Evaluation and Analysis

As we have a regression problem at hand, we will be using linear regression, decision tree and random forest algorithms on our dataset. We will be using three evaluation metrics: mean absolute error (MAE), mean squared error (MSE) and r-squared method. The evaluation metrics will help us to decide which model performs best on our dataset. The lower the MAE and MSE values, the better the model has performed. In the case of the r-squared method, a value closer to 1 means that model has fitted better.

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

dataset_normalized_copy = dataset_normalized.copy(deep=True)
not_null_rows = dataset_normalized_copy[dataset_normalized_copy.notna().all(axis=1)]
```

```
# Assign the features and target
X = not_null_rows.drop(columns=['price'])
y = not_null_rows['price']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

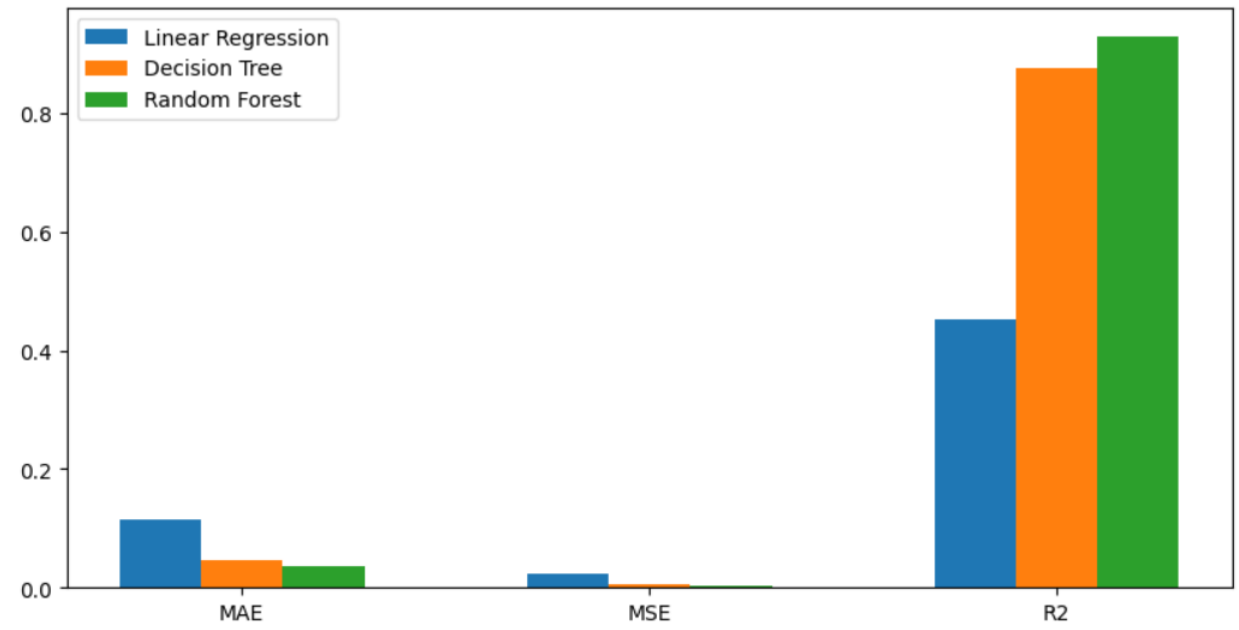
```
# Instantiate the models
linear_regression = LinearRegression()
decision_tree = DecisionTreeRegressor()
random_forest = RandomForestRegressor()

# Train the models
linear_regression.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

# Predict on the test set
linear_predictions = linear_regression.predict(X_test)
dt_predictions = decision_tree.predict(X_test)
rf_predictions = random_forest.predict(X_test)
```

```
# Evaluate the models
linear_mae = mean_absolute_error(y_test, linear_predictions)
linear_mse = mean_squared_error(y_test, linear_predictions)
linear_r2 = r2_score(y_test, linear_predictions)

dt_mae = mean_absolute_error(y_test, dt_predictions)
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_r2 = r2_score(y_test, dt_predictions)

rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
```

```
# Compare the results
print(f'Linear Regression: MAE {linear_mae}, MSE {linear_mse}, R2 {linear_r2}')
print(f'Decision Tree: MAE {dt_mae}, MSE {dt_mse}, R2 {dt_r2}')
print(f'Random Forest: MAE {rf_mae}, MSE {rf_mse}, R2 {rf_r2}')

# Create a list of the evaluation metrics
metrics = ['MAE', 'MSE', 'R2']

# Create a list of the evaluation scores for each model
lin_reg_scores = [linear_mae, linear_mse, linear_r2]
dt_scores = [dt_mae, dt_mse, dt_r2]
rf_scores = [rf_mae, rf_mse, rf_r2]

# Create the bar plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111)

# Set the bar width
bar_width = 0.2

# Set the position of the bars on the x-axis
bar_pos = np.arange(len(metrics))

# Create the bars
ax.bar(bar_pos, lin_reg_scores, bar_width, label='Linear Regression')
ax.bar(bar_pos + bar_width, dt_scores, bar_width, label='Decision Tree')
ax.bar(bar_pos + 2 * bar_width, rf_scores, bar_width, label='Random Forest')
```

```
# Set the x-axis labels
ax.set_xticks(bar_pos + bar_width)
ax.set_xticklabels(metrics)

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

```
Linear Regression: MAE 0.11426127719455725, MSE 0.02329654415313562, R2 0.45307434724353846
Decision Tree: MAE 0.04605737113595398, MSE 0.005257755090641218, R2 0.876565334494236
Random Forest: MAE 0.03539420837082417, MSE 0.002978151536772262, R2 0.9300828714100243
```



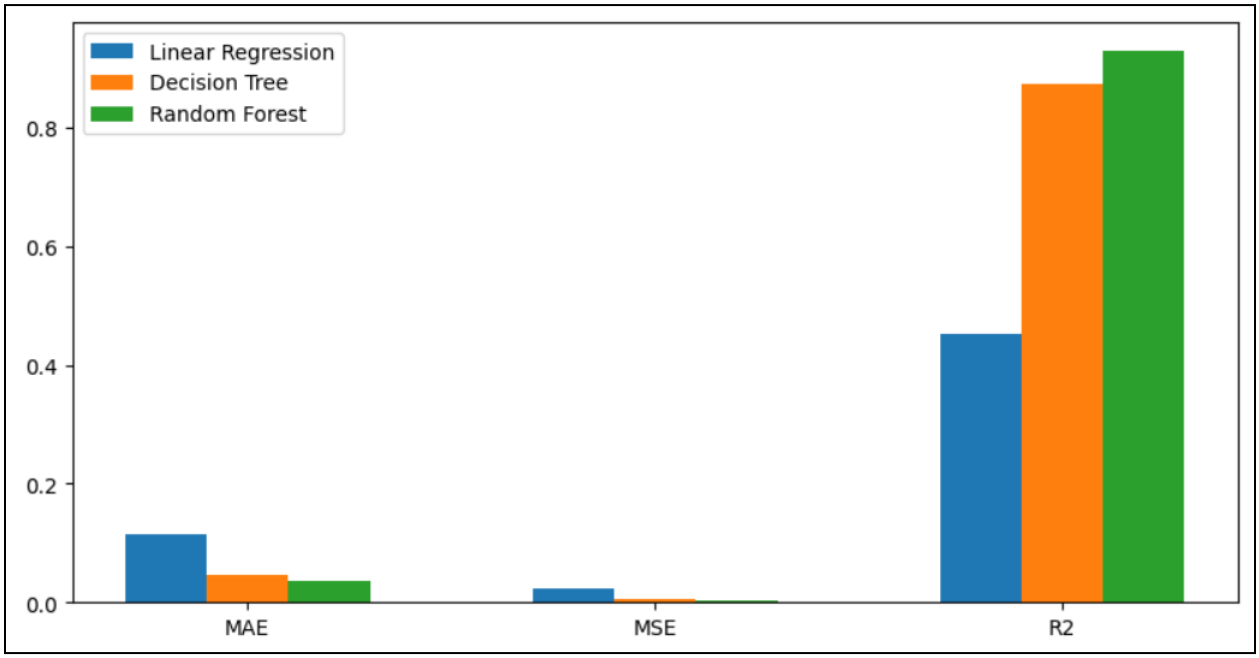

These were the results of the three models after the first pass. As expected, Decision Tree and Random Forest performed better than Linear Regression. Random Forest performed the best across the board.

To make sure that our model was not overfitting, we decided to do k-fold cross validation with the three models. We used 5 as the value for k. The following results were established:

```python
from sklearn.model_selection import cross_validate

# Linear Regression
lin_reg = LinearRegression()
lin_reg_cv_results = cross_validate(lin_reg, X, y, cv=5, scoring=['neg_mean_absolute_error', 'neg_mean_squared_error', 'r2'])
lin_reg_mae = -1 * lin_reg_cv_results['test_neg_mean_absolute_error'].mean()
lin_reg_mse = -1 * lin_reg_cv_results['test_neg_mean_squared_error'].mean()
lin_reg_r2 = lin_reg_cv_results['test_r2'].mean()

# Decision Tree
dt = DecisionTreeRegressor()
dt_cv_results = cross_validate(dt, X, y, cv=5, scoring=['neg_mean_absolute_error', 'neg_mean_squared_error', 'r2'])
dt_mae = -1 * dt_cv_results['test_neg_mean_absolute_error'].mean()
dt_mse = -1 * dt_cv_results['test_neg_mean_squared_error'].mean()
dt_r2 = dt_cv_results['test_r2'].mean()

# Random Forest
rf = RandomForestRegressor()
rf_cv_results = cross_validate(rf, X, y, cv=5, scoring=['neg_mean_absolute_error', 'neg_mean_squared_error', 'r2'])
rf_mae = -1 * rf_cv_results['test_neg_mean_absolute_error'].mean()
rf_mse = -1 * rf_cv_results['test_neg_mean_squared_error'].mean()
rf_r2 = rf_cv_results['test_r2'].mean()

# Compare the performance of the models
print(f'Linear Regression: MAE {lin_reg_mae}, MSE {lin_reg_mse}, R2 {lin_reg_r2}')
print(f'Decision Tree: MAE {dt_mae}, MSE {dt_mse}, R2 {dt_r2}')
print(f'Random Forest: MAE {rf_mae}, MSE {rf_mse}, R2 {rf_r2}')
```

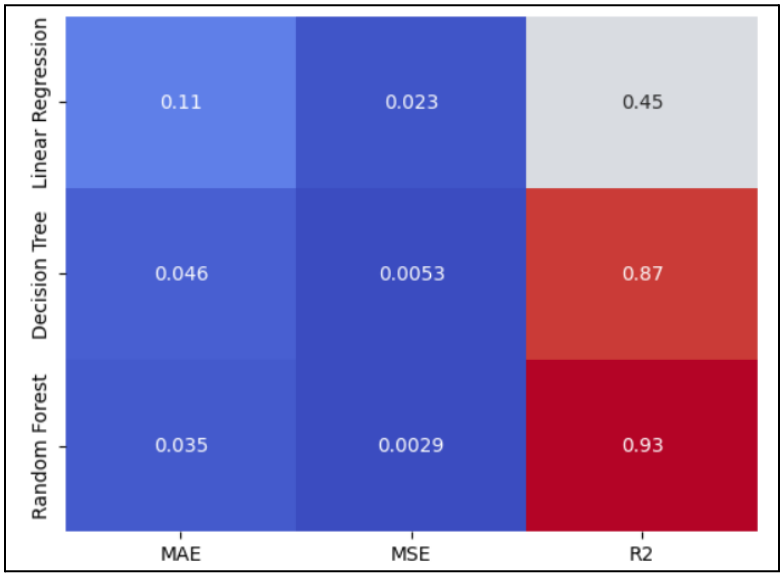
```
Linear Regression: MAE 0.11376019593810409, MSE 0.023146612458495152, R2 0.4529275564032213
Decision Tree: MAE 0.046262856024594415, MSE 0.00534307074606794, R2 0.8737016502991535
Random Forest: MAE 0.035150470846734234, MSE 0.0029252455900748444, R2 0.9308587635879672
```

```python
import matplotlib.pyplot as plt

# Create a list of the evaluation metrics
metrics = ['MAE', 'MSE', 'R2']

# Create a list of the evaluation scores for each model
lin_reg_scores = [lin_reg_mae, lin_reg_mse, lin_reg_r2]
dt_scores = [dt_mae, dt_mse, dt_r2]
rf_scores = [rf_mae, rf_mse, rf_r2]

# Create the bar plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111)

# Set the bar width
bar_width = 0.2

# Set the position of the bars on the x-axis
bar_pos = np.arange(len(metrics))

# Create the bars
ax.bar(bar_pos, lin_reg_scores, bar_width, label='Linear Regression')
ax.bar(bar_pos + bar_width, dt_scores, bar_width, label='Decision Tree')
ax.bar(bar_pos + 2 * bar_width, rf_scores, bar_width, label='Random Forest')

# Set the x-axis labels
ax.set_xticks(bar_pos + bar_width)
ax.set_xticklabels(metrics)

# Add the legend
ax.legend()

# Show the plot
plt.show()
```




Only minor differences were noticed in the results, hence, we can be confident that our model does not overfit and neither does it underfit. Here is a heatmap that also shows the same results as above from the cross validation run.

```python
# Create a dataframe that contains the evaluation scores for each model and metric
scores_df = pd.DataFrame({'MAE': [lin_reg_mae, dt_mae, rf_mae],
                          'MSE': [lin_reg_mse, dt_mse, rf_mse],
                          'R2': [lin_reg_r2, dt_r2, rf_r2]},
                         index=['Linear Regression', 'Decision Tree', 'Random Forest'])

# Create the heatmap
sns.heatmap(scores_df, annot=True, cmap='coolwarm', cbar=False)

# Show the plot
plt.show()
```

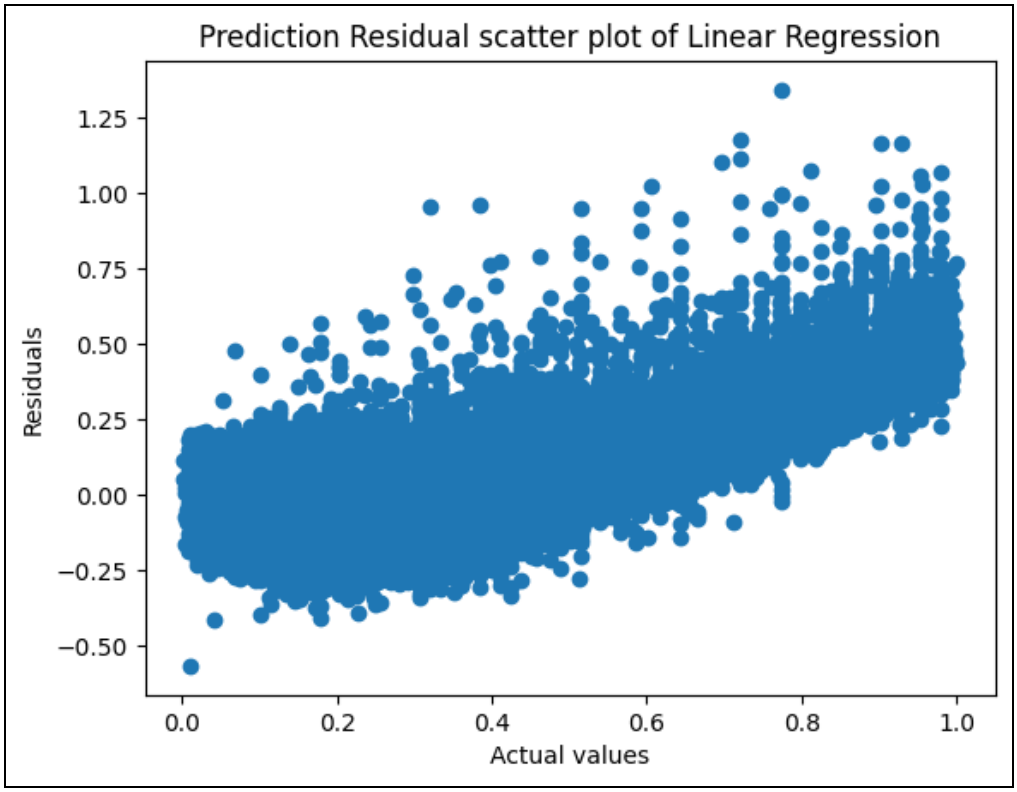


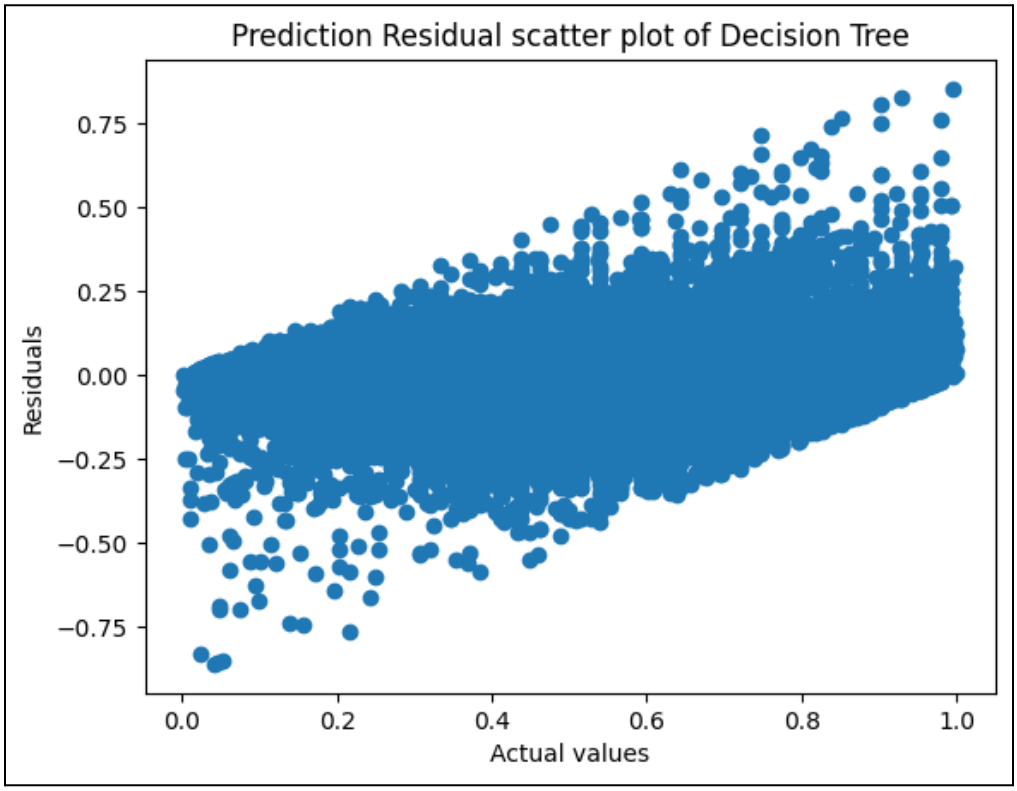

According to the scores, Random Forest works the best for our dataset. Second is Decision Tree and last is Linear Regression. Random Forest and Decision Tree perform better than Linear Regression because they are able to find non-linear relationships between the features and the target variable.

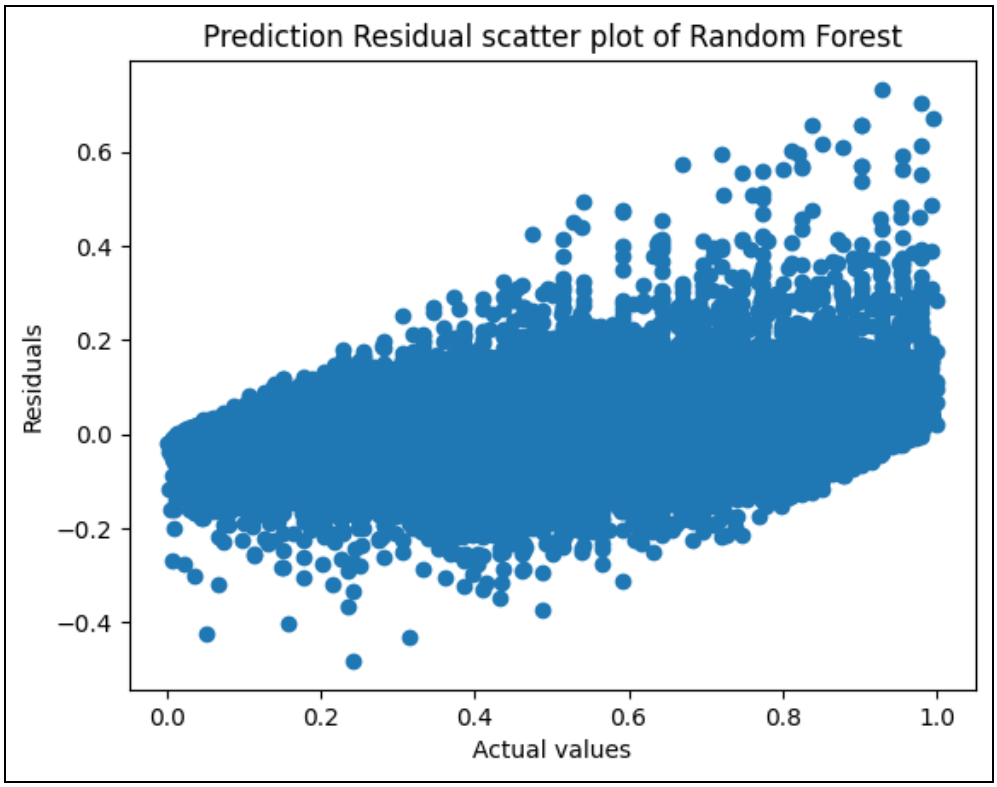
Here are some prediction residual scatter plots for each model:

```python
def prediction_residual_plotter(y_test, y_pred, model_name):
    residuals = y_test - y_pred

    # Plot the residuals
    plt.scatter(y_test, residuals)
    plt.xlabel('Actual values')
    plt.ylabel('Residuals')
    plt.title('Prediction Residual scatter plot of ' + model_name)
    plt.show()
```

```python
prediction_residual_plotter(y_test, linear_predictions, 'Linear Regression')
prediction_residual_plotter(y_test, dt_predictions, 'Decision Tree')
prediction_residual_plotter(y_test, rf_predictions, 'Random Forest')
```


Prediction Residual scatter plot of Linear Regression
Residuals
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0.0
0.2
0.4
0.6
0.8
1.0
Actual values


Prediction Residual scatter plot of Decision Tree
Residuals
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0.0
0.2
0.4
0.6
0.8
1.0
Actual values

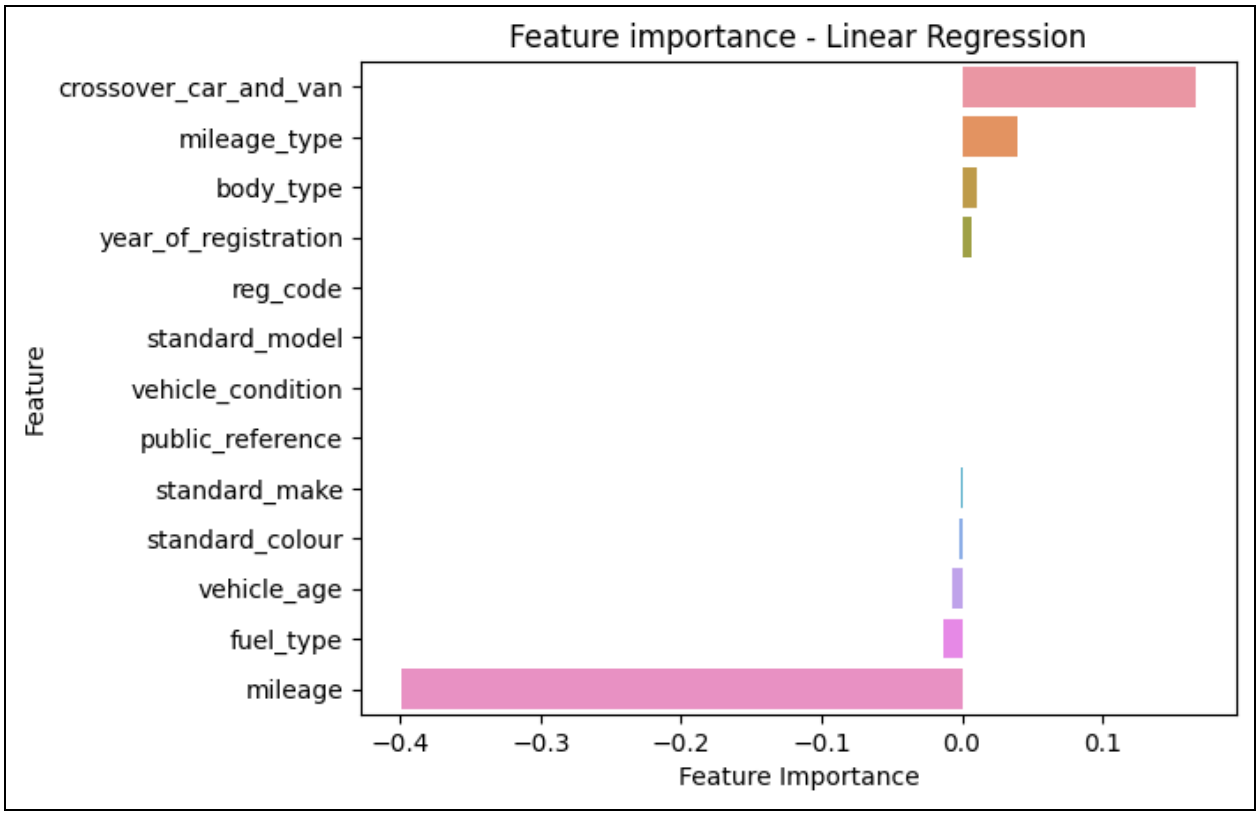


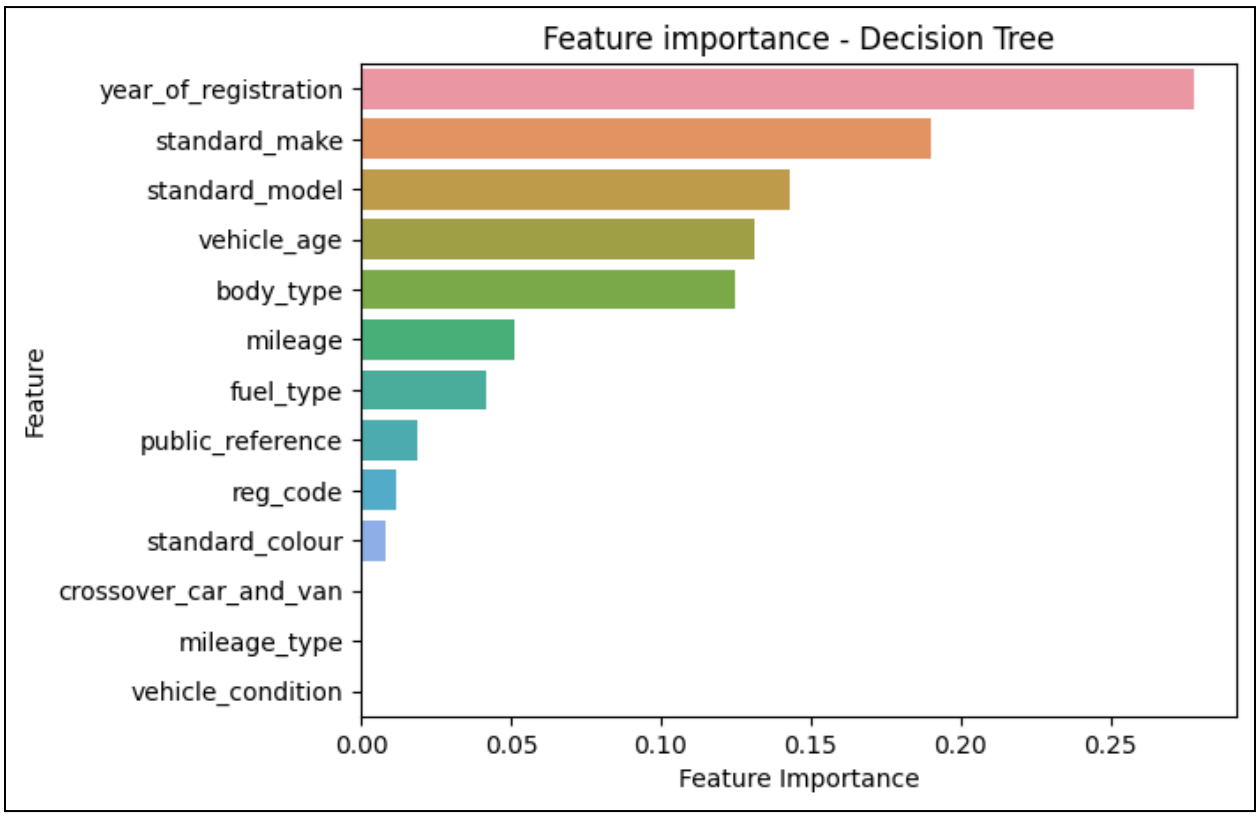

## Feature Importance according to Models Used

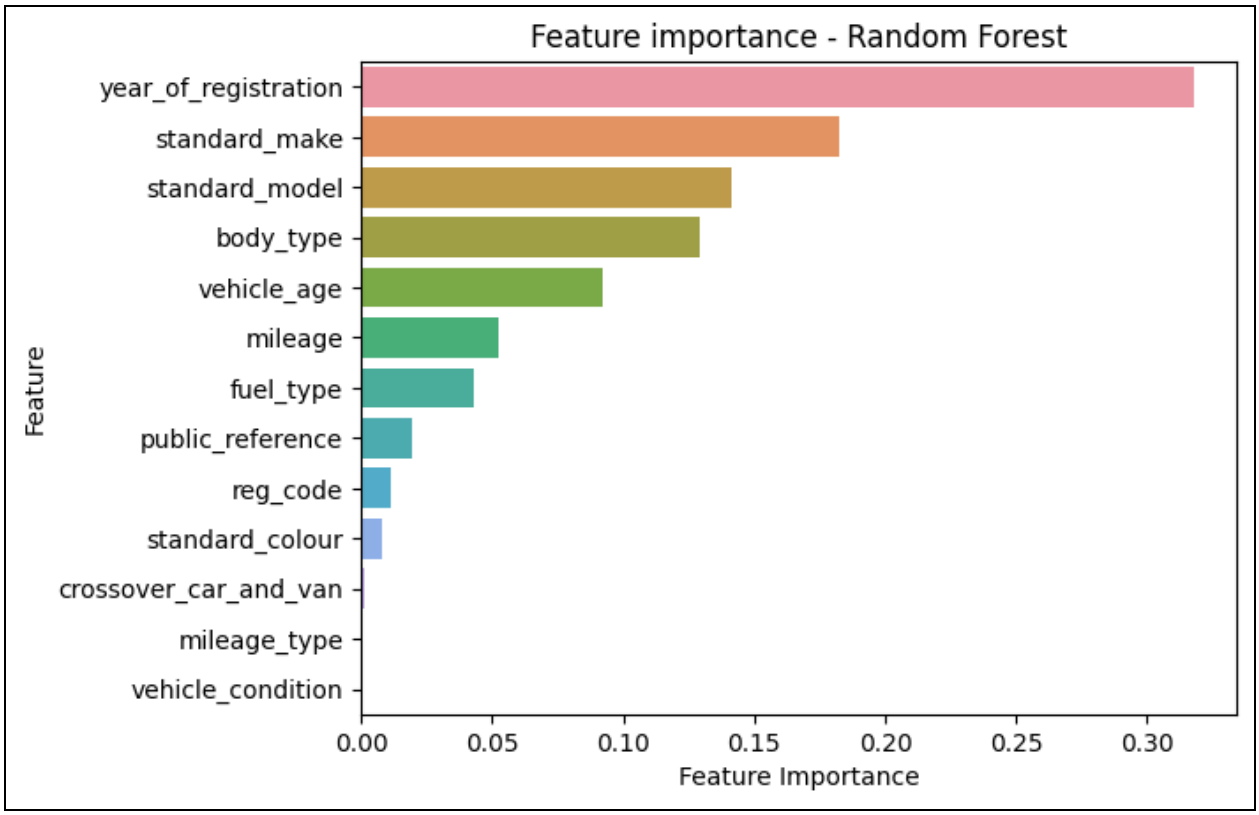
Decision Tree and Random Forest were the best performers among the three tested models. Feature importance plots for each model give an idea about the features which were given more weight when predicting target variables.

Decision Tree and Random Forest had features in the same order except for their importance value was different. Year of registration played a vital role in both cases. Mileage was the most important feature for the Linear Regression model.

Feature importance of these models can give you an idea about which features are the better options to predict the price of a vehicle.


Feature importance - Linear Regression
crossover_car_and_van
mileage_type
body_type
year_of_registration
reg_code
standard_model
vehicle_condition
public_reference
standard_make
standard_colour
vehicle_age
fuel_type
mileage
Feature
−0.4
−0.3
−0.2
−0.1
0.0
0.1
Feature Importance


Feature importance - Decision Tree
year_of_registration
standard_make
standard_model
vehicle_age
body_type
mileage
fuel_type
public_reference
reg_code
standard_colour
crossover_car_and_van
mileage_type
vehicle_condition
Feature
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance




Residual scatter plots are a great way to do fine grained analysis. It allows you to look at specific points that lie far away from line of best of fit. As you can see from the scatter plots above, all the points have been plotted. We can look at edge cases by putting up limits and then looking at each specific case to notice if there is a pattern in the points that were off from the line of best fit. This could help to improve the model that we are using.