A1 - Predicting Car Prices

Importing the libraries

In [337…
```
#Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn import preprocessing
warnings.filterwarnings('ignore')
```

Loading data

In [338…
```
# Load the dataset using pandas and storing them in a dataframe variable
df = pd.read_csv('Cars.csv')
```

In [339…
```
# Displaying the first few rows of the dataframe
df.head()
```

Out[339…

	name	year	selling_price	km_driven	fuel	seller_type	transmission	ow
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	F Ow
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Seco Ow
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Th Ow
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	F Ow
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	F Ow

In [340…
```
#Checking the statistical summary of the dataframe
df.describe()
```

Out[340…

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

In [341…

```
#Checking the data types and non-null values in the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [342…

```
#Checking the number of rows and columns in the dataframe
df.shape
```

Out[342…

```
(8128, 13)
```

In [343…

```
#Checking the column names in the dataframe
df.columns
```

Out[343…

```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_typ
e',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [344…

```
#Checking for unique values in "owner" column
df['owner'].unique()
```

Out[344…
```
array(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'], dtype=object)
```

Task 1: Preparing the datasets

Changing the features

In [345…
```
#Mapping the owner column to numerical values.
ownermap = {
    "First Owner":1,
    "Second Owner":2,
    "Third Owner":3,
    "Fourth & Above Owner":4,
    "Test Drive Car":5
    }
```

In [346…
```
#Checking if the owner column exists in the dataframe and then mapping it
if 'owner' in df.columns:
    df['owner'] = df['owner'].map(ownermap)
```

In [347…
```
#Checking if mapping is done correctly
df.owner.unique()
```

Out[347…
```
array([1, 2, 3, 4, 5])
```

Removing the rows of 'Fuel' column with values of LPG and CNG

In [348…
```
#Getting the count of unique values in the 'Fuel' column
df['fuel'].unique()
```

Out[348…
```
array(['Diesel', 'Petrol', 'LPG', 'CNG'], dtype=object)
```

In [349…
```
#Storing the unique values to remove in an array
fuel_to_remove = ['CNG', 'LPG']
```

In [350…
```
#Removing the unwanted fuel types from the dataframe
df = df[~df['fuel'].isin(fuel_to_remove)]
```

In [351…
```
#checking if the unwanted fuel types are removed
df['fuel'].unique()
```

Out[351…
```
array(['Diesel', 'Petrol'], dtype=object)
```

Removing "kmpl" and converting the column to numerical type for feature mileage

In [352…
```python
#Getting the values of mileage column
df['mileage'].head()
```

Out[352…
```
0     23.4 kmpl
1    21.14 kmpl
2     17.7 kmpl
3     23.0 kmpl
4     16.1 kmpl
Name: mileage, dtype: object
```

In [353…
```python
#Removing the "kmpl" from the mileage column
df['mileage']= df['mileage'].str.split( ).str[0]
```

In [354…
```python
#Converting the mileage column to float type
df['mileage'] = df['mileage'].astype(float)
```

In [355…
```python
#Checking if the conversation is done correctly
df['mileage'].dtype
```

Out[355…
```
dtype('float64')
```

Removing "CC" and converting the column to numerical type for feature engine

In [356…
```python
#Getting the values of 'engine' feature column
df['engine'].head()
```

Out[356…
```
0    1248 CC
1    1498 CC
2    1497 CC
3    1396 CC
4    1298 CC
Name: engine, dtype: object
```

In [357…
```python
#Removing the 'CC' from the engine column using str.split() method
df['engine'] = df['engine'].str.split( ).str[0]
```

In [358…
```python
#Verifying the changes in the engine column
df['engine'].head()
```

Out[358…
```
0    1248
1    1498
2    1497
3    1396
4    1298
Name: engine, dtype: object
```

In [359…
```python
#converting the data type of engine column to float
df['engine'] = df['engine'].astype(float)
```

In [360…
```python
#verifying the data type of engine column
df['engine'].dtype
```

Out[360…
```
dtype('float64')
```

Removing "bhp" and converting the column to numerical type for feature max_power

In [361…
```
#Getting the values of the column 'max_power'
df['max_power'].head()
```

Out[361…
```
0        74 bhp
1    103.52 bhp
2        78 bhp
3        90 bhp
4      88.2 bhp
Name: max_power, dtype: object
```

In [362…
```
#Removing the 'bhp' from the max_power column using str.split() method
df['max_power'] = df['max_power'].str.split( ).str[0]
```

In [363…
```
#Converting the data type of max_power column to float
df['max_power'] = df['max_power'].astype(float)
```

In [364…
```
#Verifying the changes
df['engine'].head()
```

Out[364…
```
0    1248.0
1    1498.0
2    1497.0
3    1396.0
4    1298.0
Name: engine, dtype: float64
```

In [365…
```
#Converting the mileage column to float type
df['mileage'] = df['mileage'].astype(float)
```

In [366…
```
#Verifying the changes
df['engine'].dtypes
```

Out[366…
```
dtype('float64')
```

Taking only the first word for the feature brand

In [367…
```
df.columns
```

Out[367…
```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_typ
e',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [368…
```
#Renaming the column 'name' to 'brand'
df.rename(columns={'name':'brand'}, inplace=True)
#checking the changes
df.columns
```

Out[368…
```
Index(['brand', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_ty
pe',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [369…
```
#Getting information about the column 'brand'
df['brand'].head()
```

Out[369…
```
0          Maruti Swift Dzire VDI
1     Skoda Rapid 1.5 TDI Ambition
2         Honda City 2017-2020 EXi
3        Hyundai i20 Sportz Diesel
4          Maruti Swift VXI BSIII
Name: brand, dtype: object
```

In [370…
```
#Only taking the first word from the brand column by using the str.split(
df['brand'] = df['brand'].str.split( ).str[0]
```

In [371…
```
#Verifying the changes
df['brand'].head()
```

Out[371…
```
0     Maruti
1      Skoda
2      Honda
3    Hyundai
4     Maruti
Name: brand, dtype: object
```

Dropping the feature torque

In [372…
```
#Removing the feature 'torque' from the dataframe using the drop() method
df = df.drop(['torque'], axis=1)
```

Removing the 'Test Drive Cars' i.e. owner = 5 from the data set.

In [373…
```
#Removing the 'Owner=5' from the dataframe using the query() method
```

```
df = df.query("owner != '5'")
```

In [374…

```
#Verifying the changes on the owner column
df['owner'].unique()
```

Out[374…

```
array([1, 2, 3, 4, 5])
```

In [375…

```
#Checking the cleaned dataframe
df.head()
```

Out[375…

	brand	year	selling_price	km_driven	fuel	seller_type	transmission	owne
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	

Changing the columns 'brand', 'fuel, 'seller_type' and 'trasmission' to numerical format.

In [376…

```
#Importing the LabelEncoder from sklearn library
from sklearn.preprocessing import LabelEncoder

#Initializing the LabelEncoder
label_encoder_brand = LabelEncoder()
label_encoder_fuel = LabelEncoder()
label_encoder_seller_type = LabelEncoder()
label_encoder_transmission = LabelEncoder()

#Transforming the categorical columns using the fit_transform() method
df['brand'] = label_encoder_brand.fit_transform(df['brand'])
df['fuel'] = label_encoder_fuel.fit_transform(df['fuel'])
df['seller_type'] = label_encoder_seller_type.fit_transform(df['seller_ty
df['transmission'] = label_encoder_transmission.fit_transform(df['transmi
```

In [377…

```
#Checking for the changes made
df.head()
```

Out[377…

	brand	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	20	2014	450000	145500	0	1	1	1
1	27	2014	370000	120000	0	1	1	2
2	10	2006	158000	140000	1	1	1	3
3	11	2010	225000	127000	0	1	1	1
4	20	2007	130000	120000	1	1	1	1

In [378…
```
#Creating a csv of the encoded data set.
df.to_csv('le_cars_data.csv', sep = ',', index=False, encoding='utf-8 ')
```

Exploratory Data Analysis

In [379…
```
#Seperating the numerical and categorical columns from the dataframe
#Excluding the column 'selling_price' from the numerical columns using df

df_copy = df.loc[:, df.columns != 'selling_price']

#Seperating the numerical and categorical columns from the dataframe

num_cols = df_copy.select_dtypes(include=['int64', 'float64'])
cat_cols = df_copy.select_dtypes(exclude=['int64', 'float64'])
```

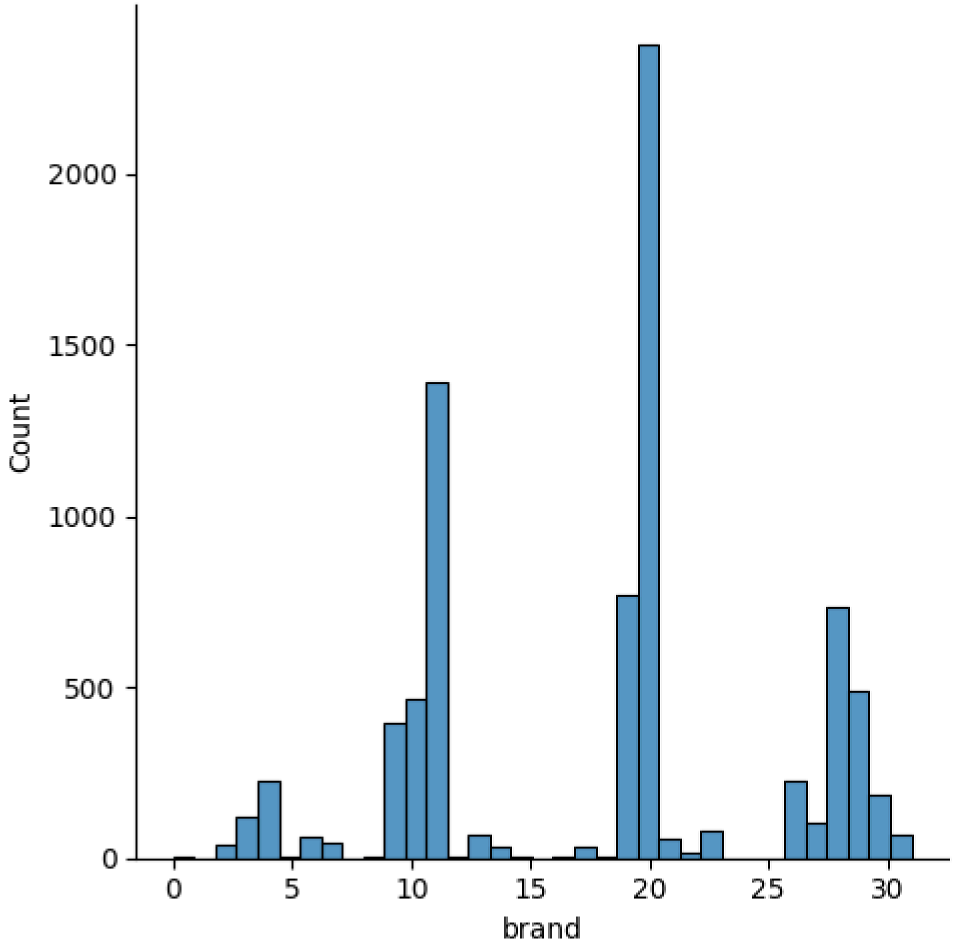
In [380…
```
#Verifying the changes made to the dataset
num_cols.columns, cat_cols.columns
```

Out[380…
```
(Index(['brand', 'year', 'km_driven', 'fuel', 'seller_type', 'transmissi
on',
        'owner', 'mileage', 'engine', 'max_power', 'seats'],
       dtype='object'),
 Index([], dtype='object'))
```

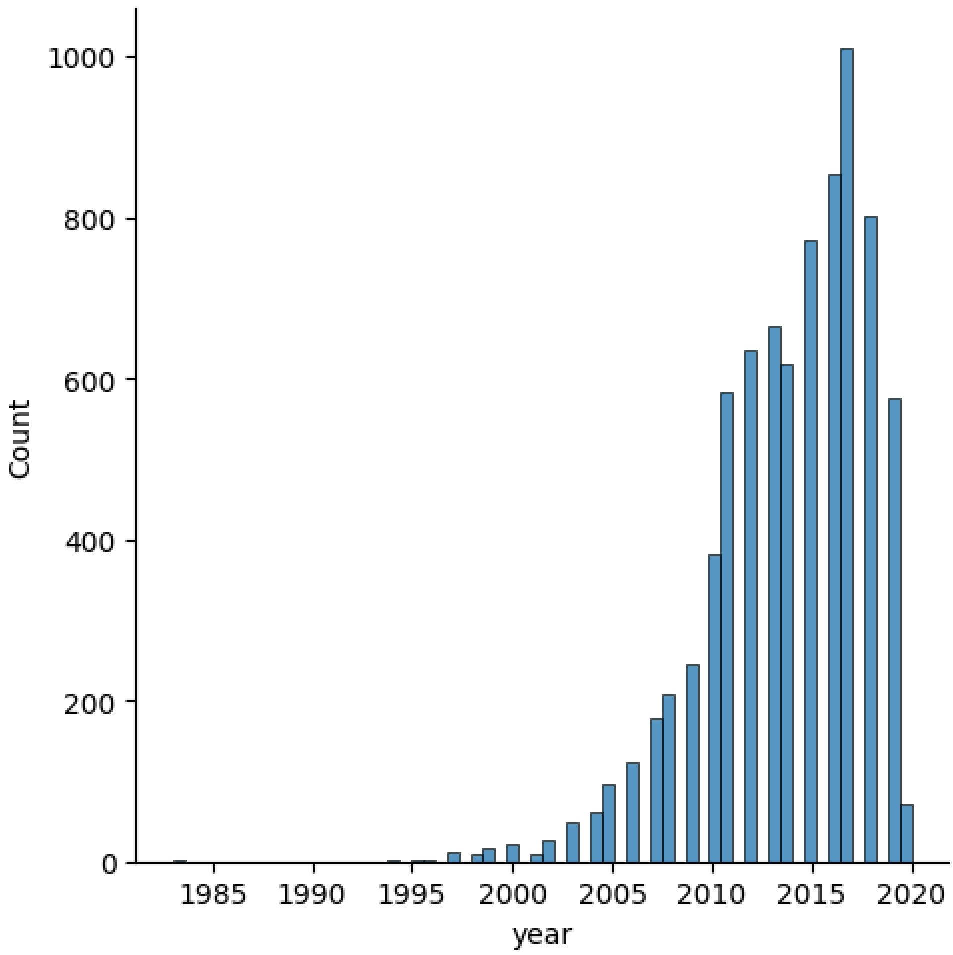
Univariate Analysis

Distribution plot

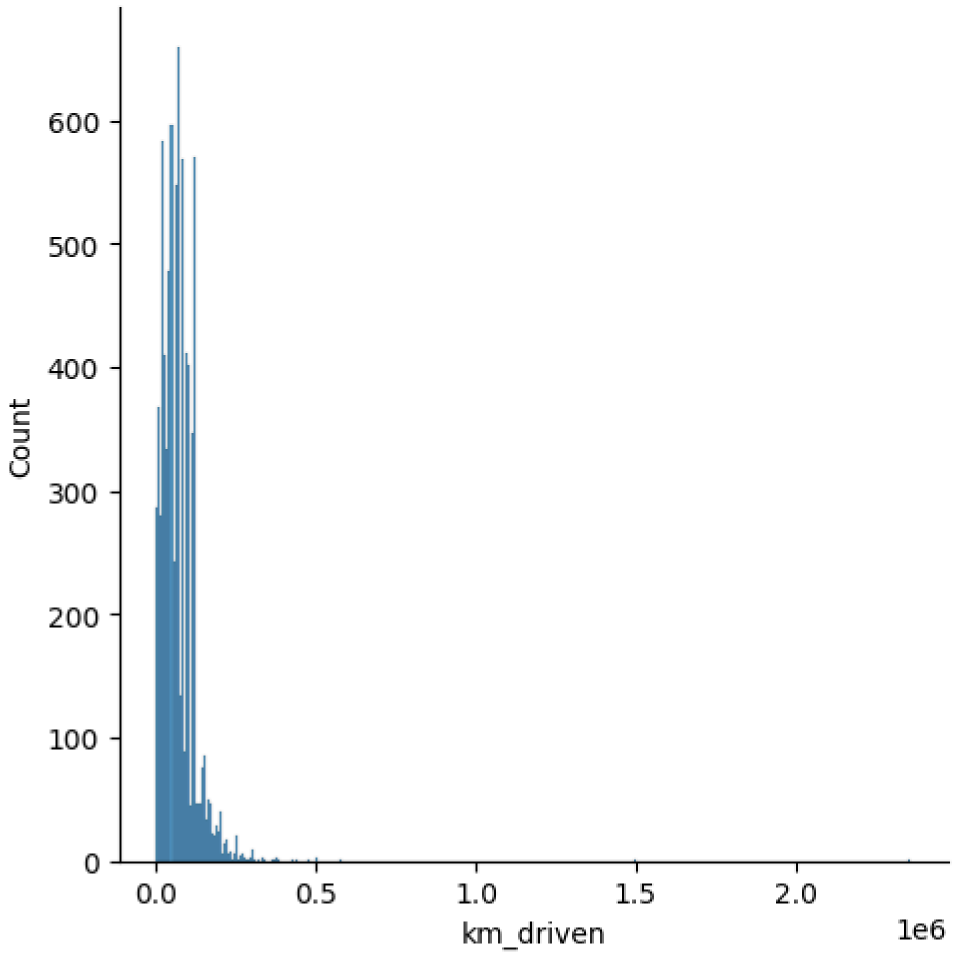
In [381…
```
#Analysing the count of all nummerical colums using the distribution plot
for col in num_cols.columns:
    sns.displot(df, x=df[col])
```

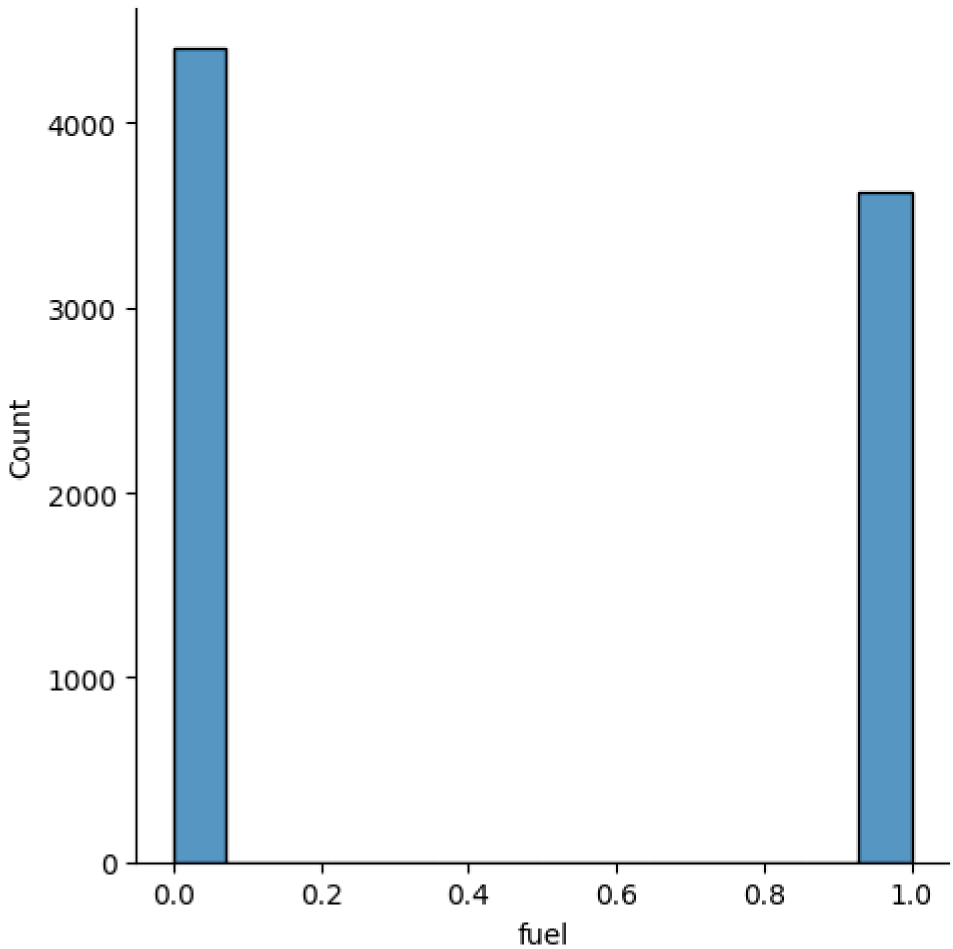


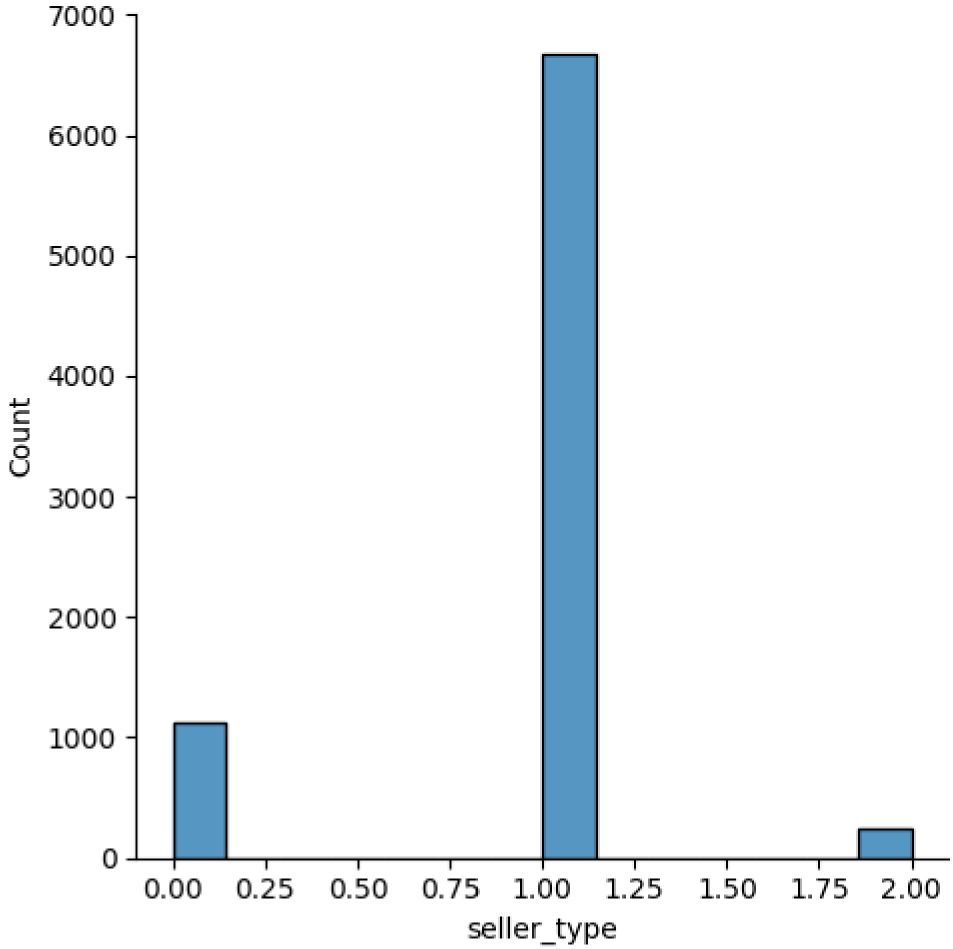
Count
brand
0
500
1000
1500
2000
0
5
10
15
20
25
30

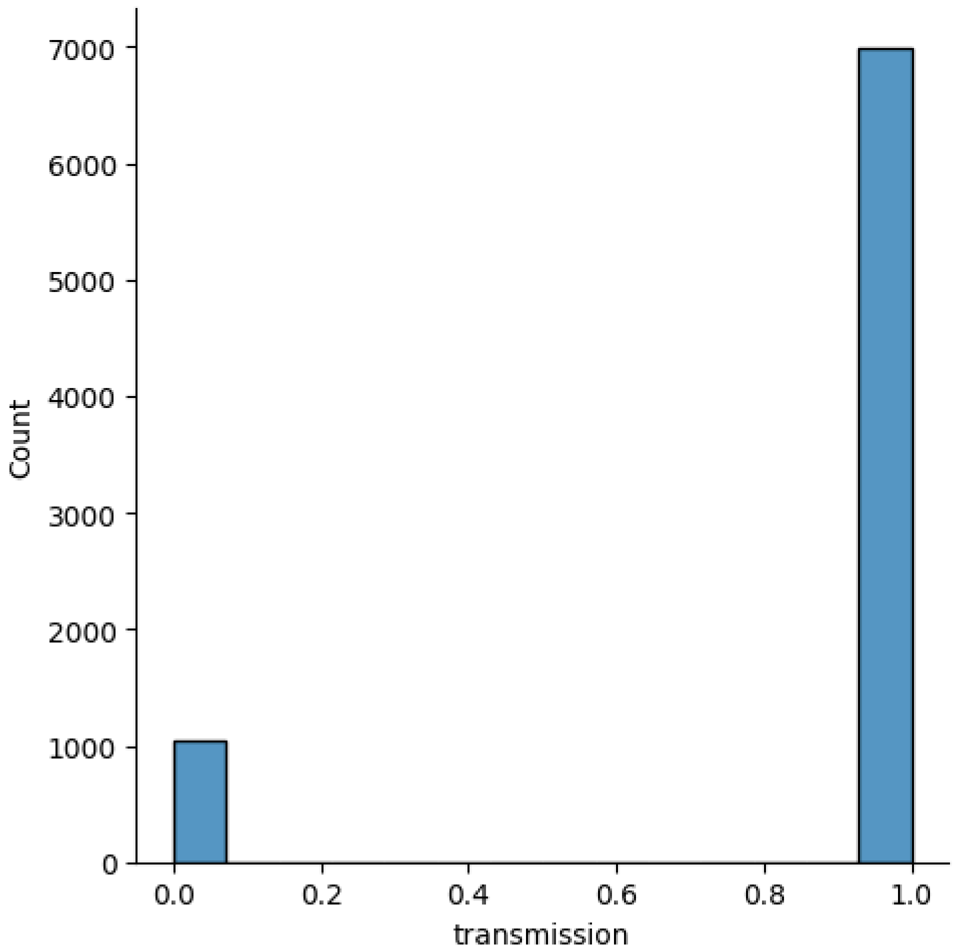


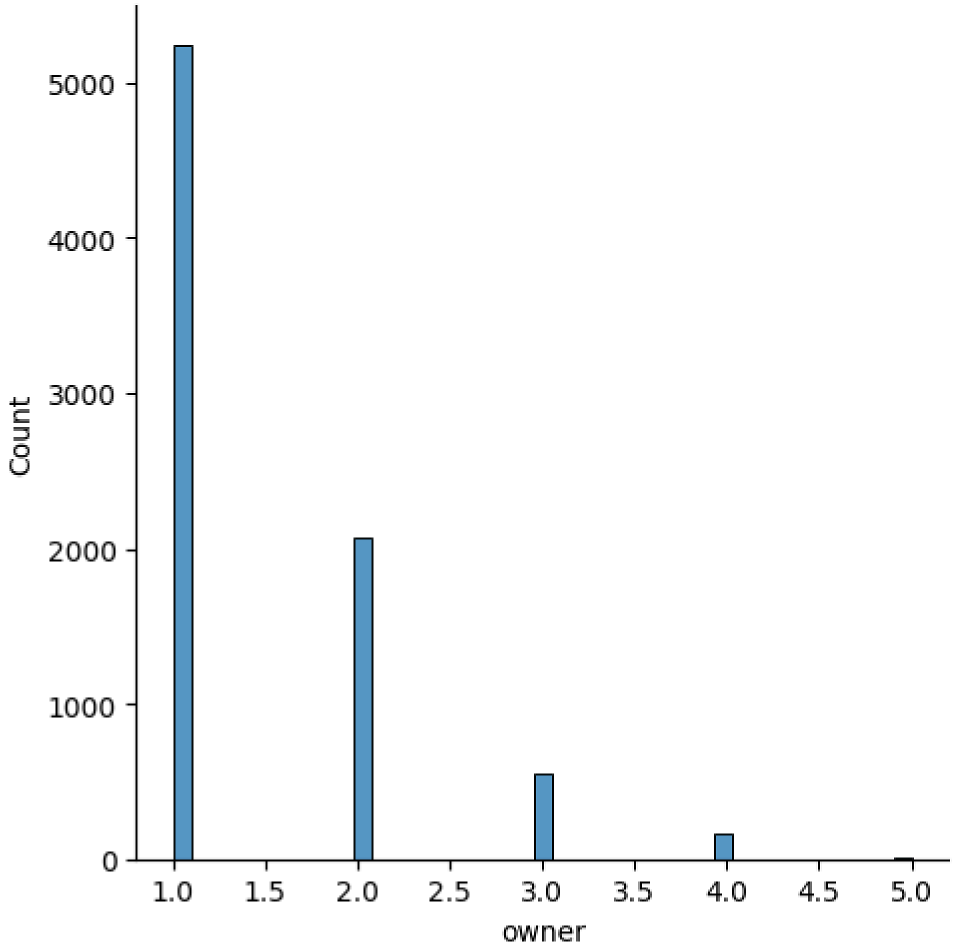
Count
year
0
200
400
600
800
1000
1985
1990
1995
2000
2005
2010
2015
2020

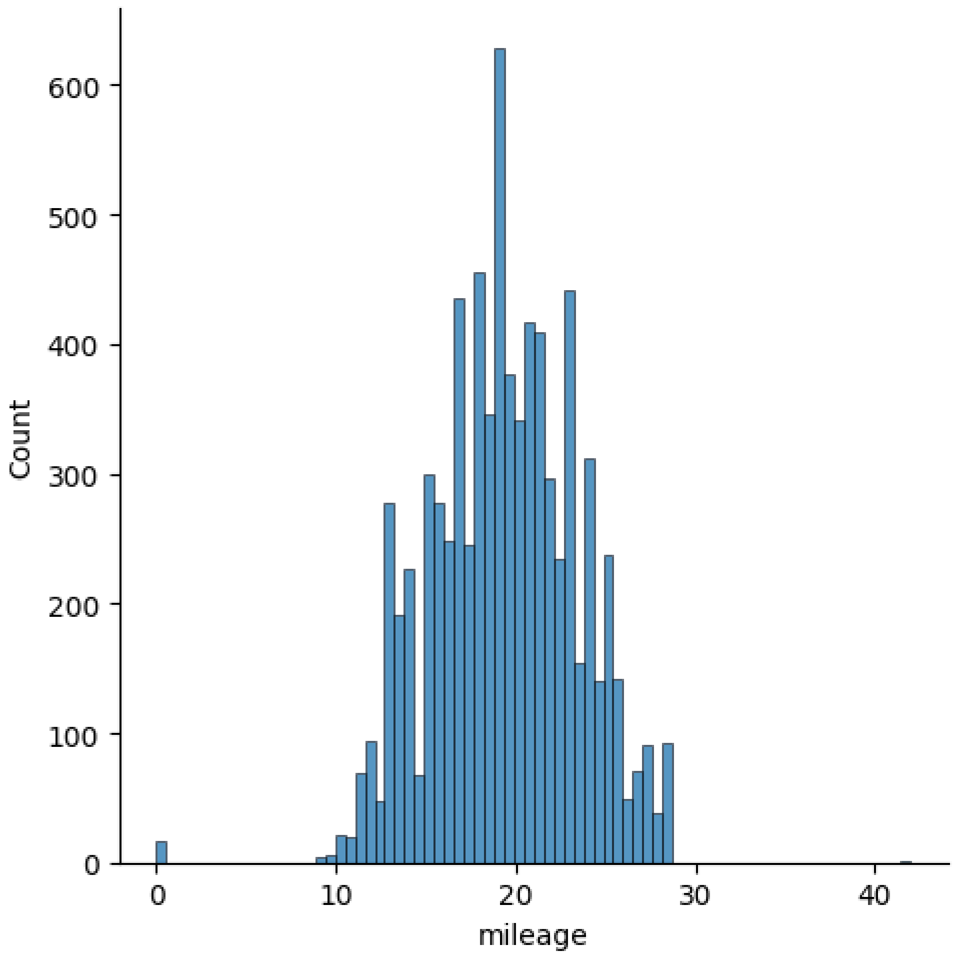

Count
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
1e6
km_driven

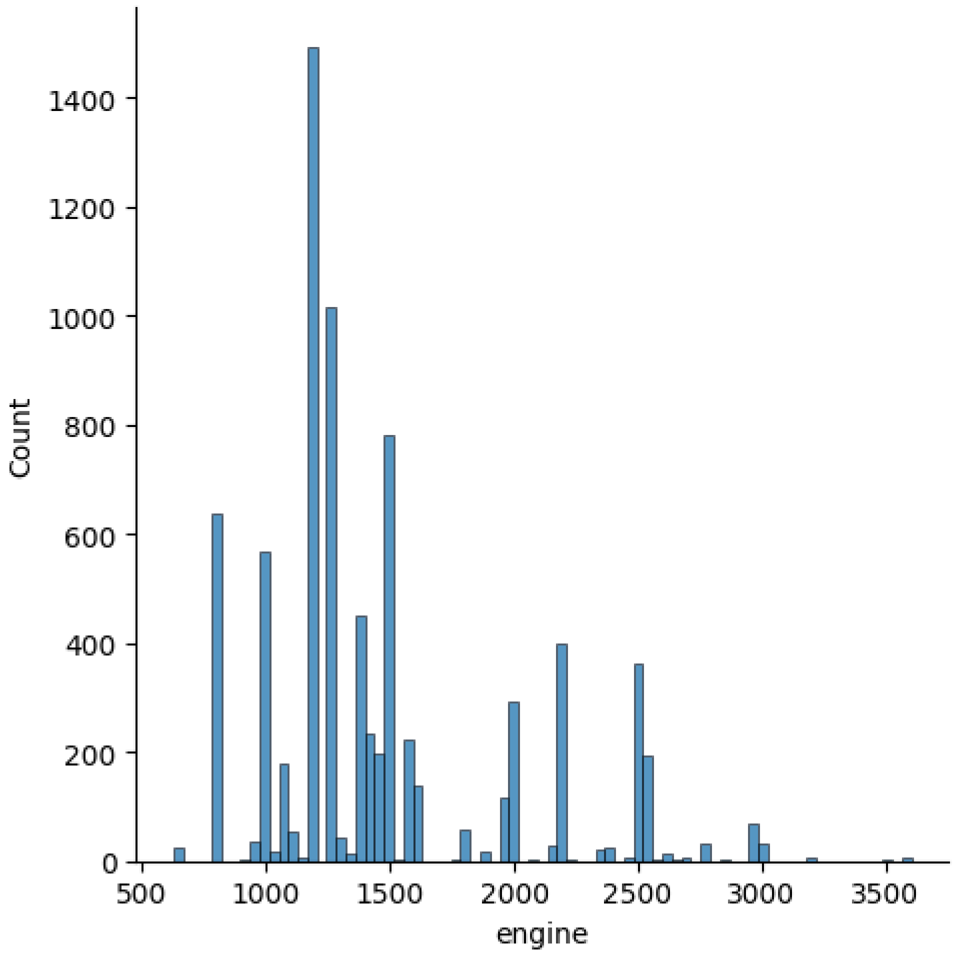

Count
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
fuel

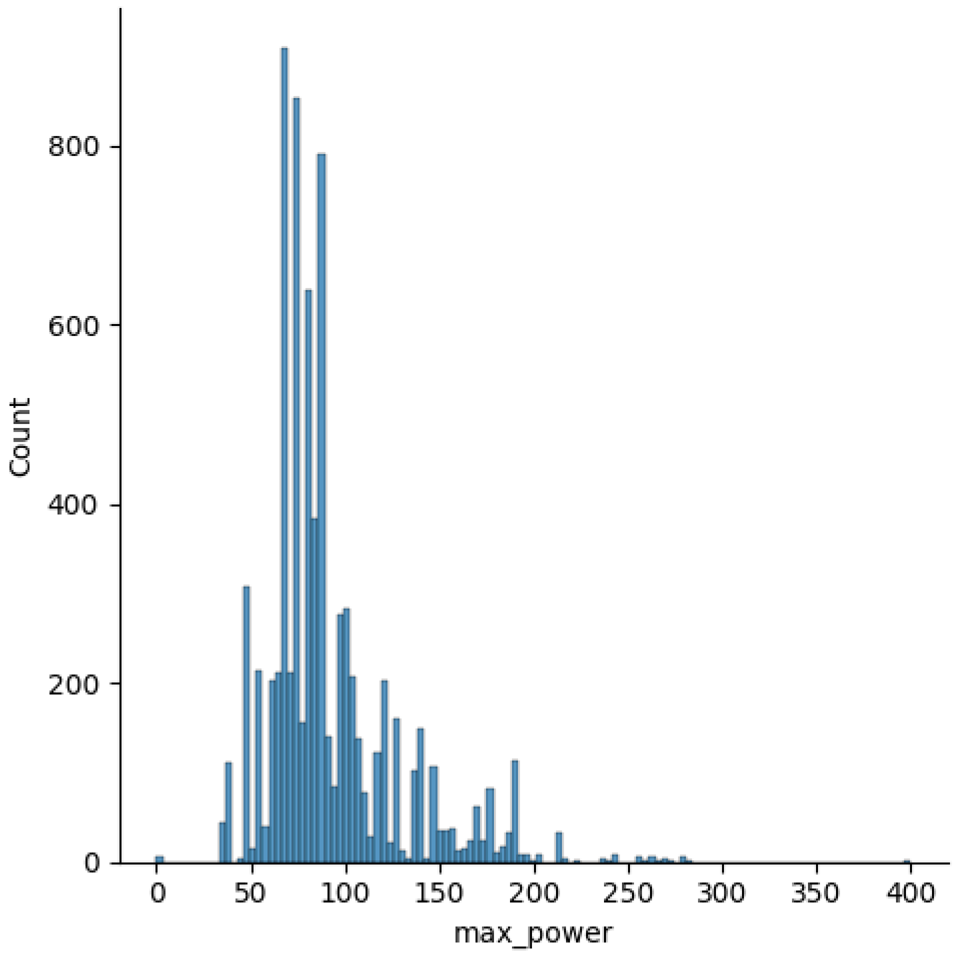

7000
6000
5000
4000
3000
2000
1000
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
seller_type

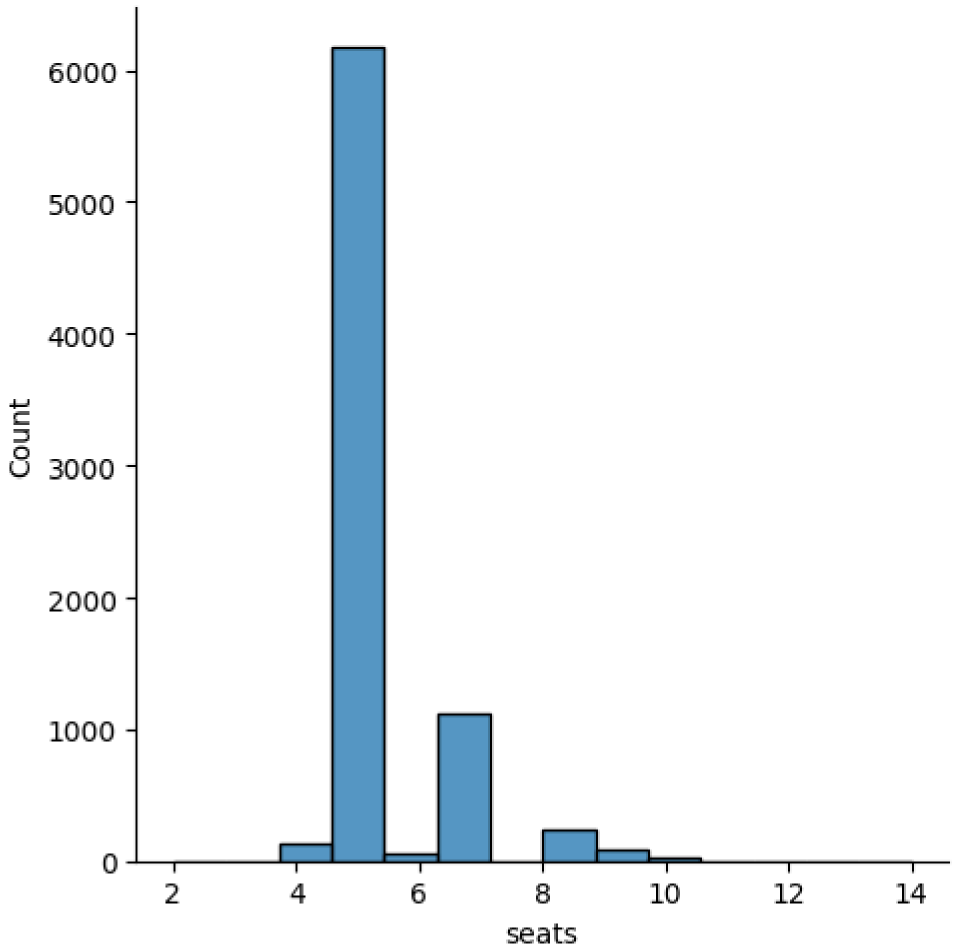

7000
6000
5000
4000
3000
2000
1000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
transmission


5000
4000
3000
2000
1000
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
owner


600
500
400
300
200
100
0
Count
0
10
20
30
40
mileage


Count
engine
0
200
400
600
800
1000
1200
1400
500
1000
1500
2000
2500
3000
3500


Count
max_power
0
200
400
600
800
0
50
100
150
200
250
300
350
400



Multivariate Analysis

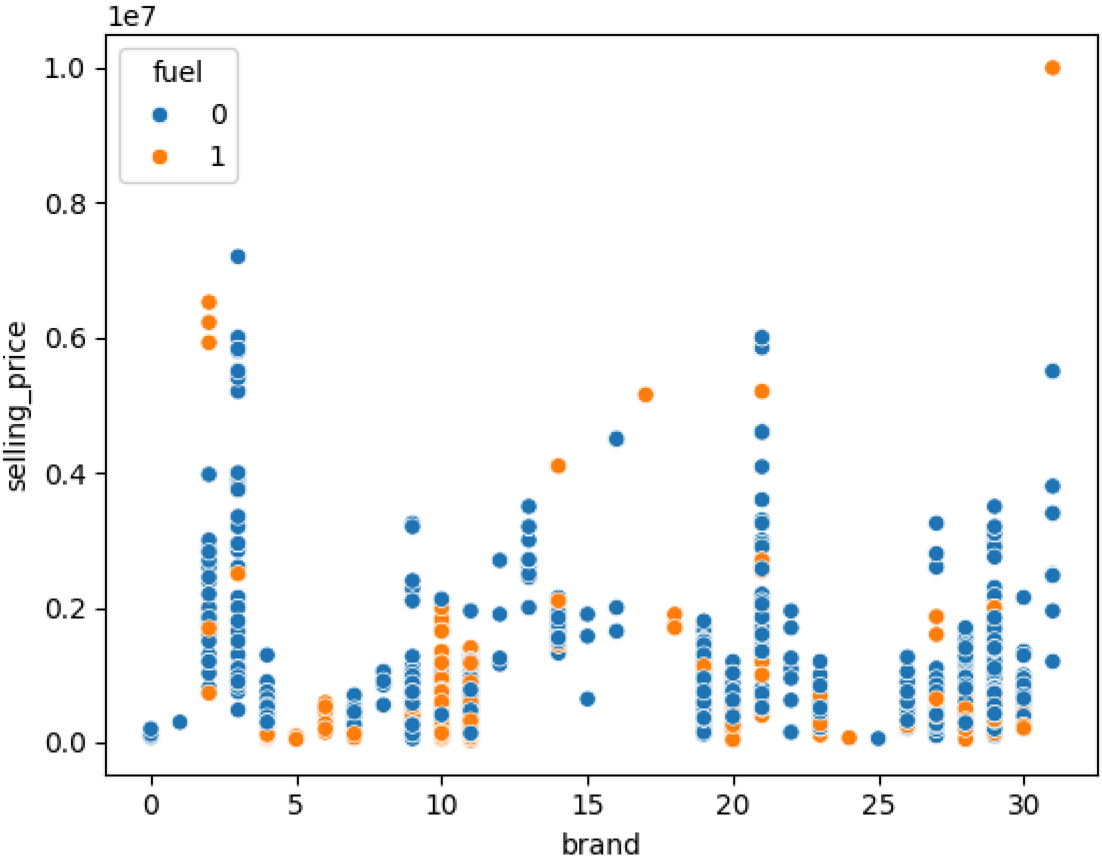
Scatterplot

In [382…

```
#Plotting the features in the scatter plot to see the relationship betwee
sns.scatterplot(x=df['brand'], y=df['selling_price'], hue=df['fuel'])
```

Out[382…

```
<Axes: xlabel='brand', ylabel='selling_price'>
```

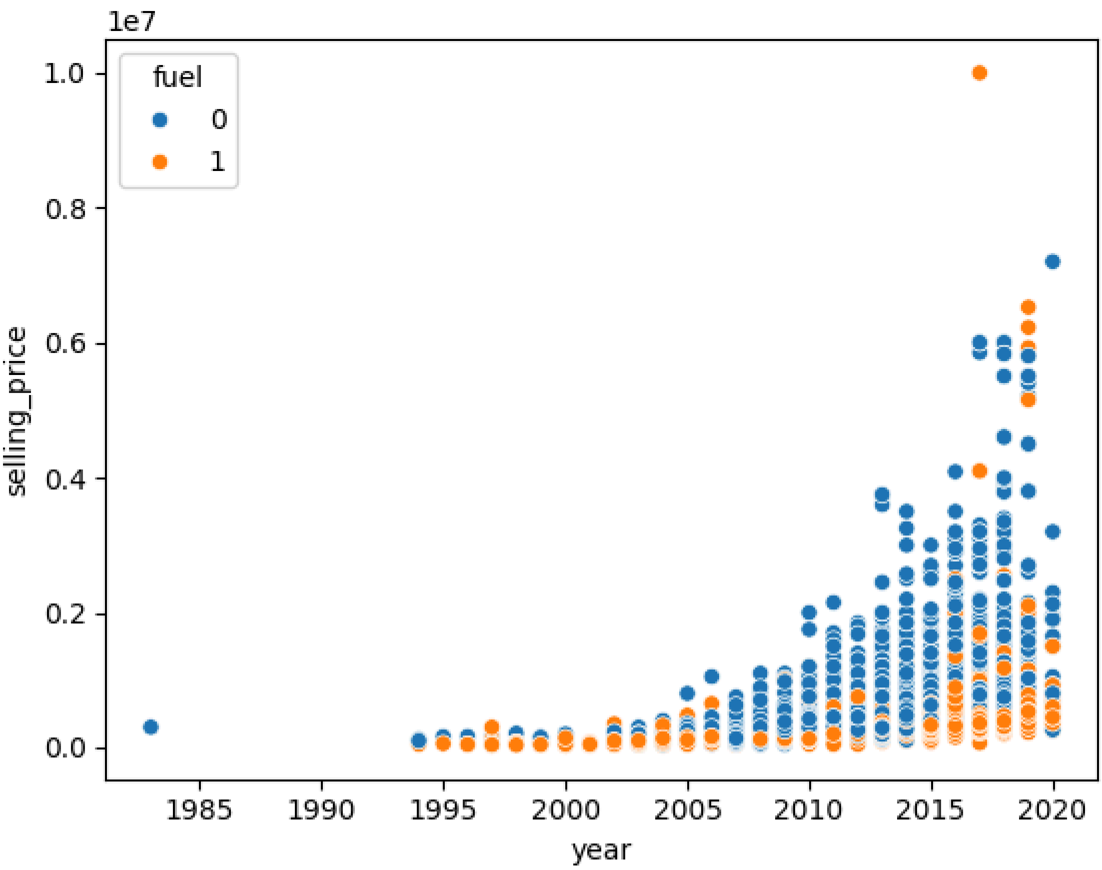


In [383…

```
sns.scatterplot(x=df['year'], y=df['selling_price'], hue=df['fuel'])
```

Out[383…

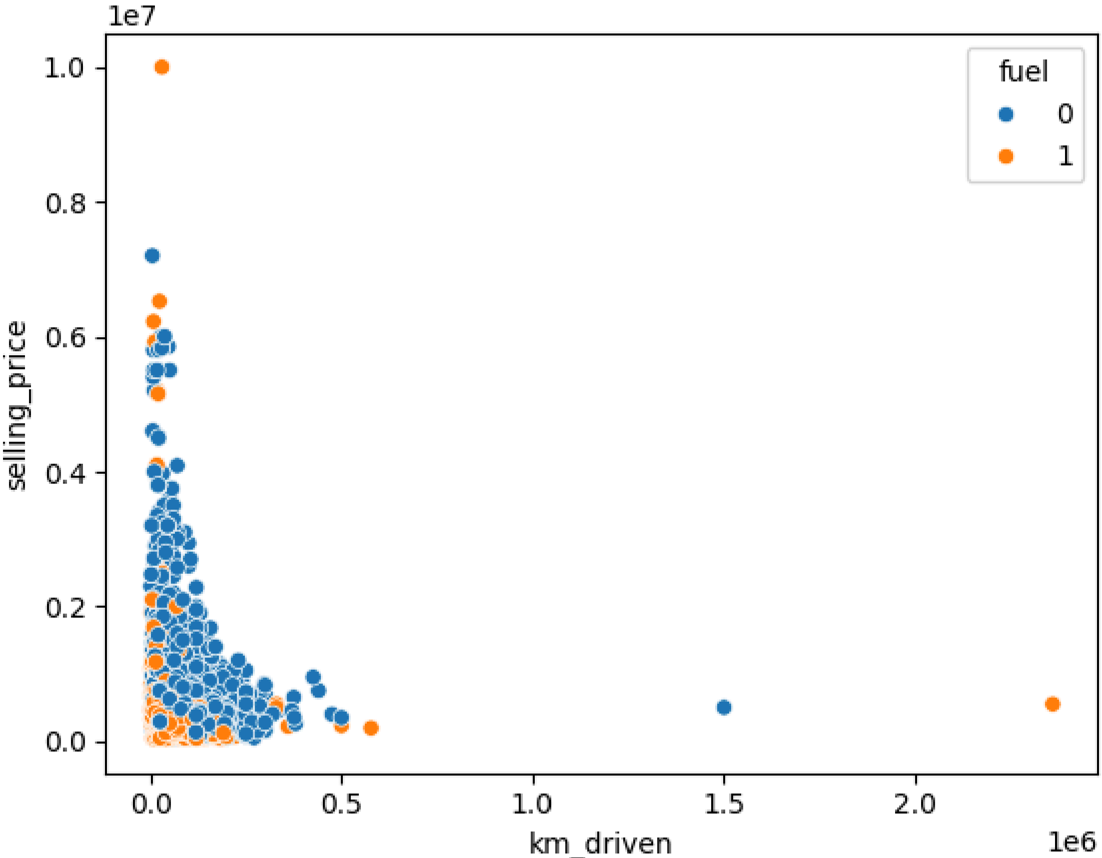
```
<Axes: xlabel='year', ylabel='selling_price'>
```



In [384…

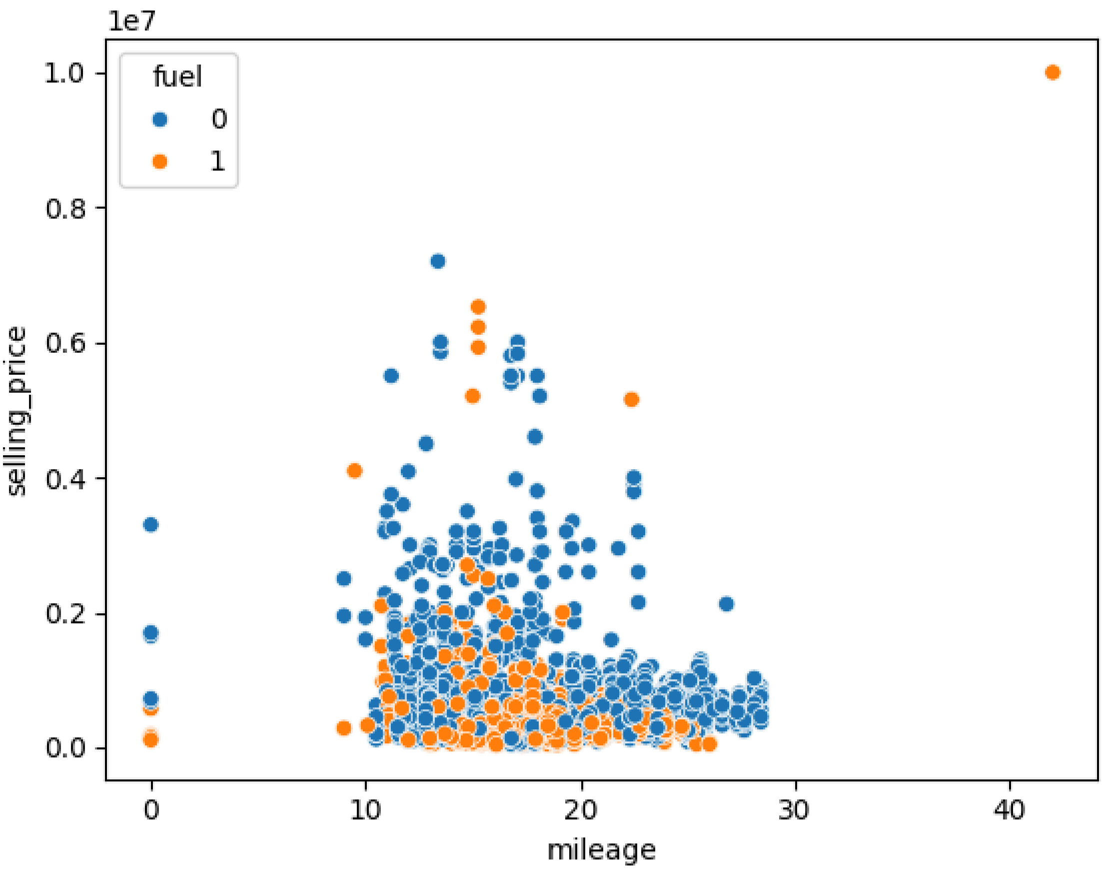
```
sns.scatterplot(x=df['km_driven'], y=df['selling_price'], hue=df['fuel'])
```

Out[384… <Axes: xlabel='km_driven', ylabel='selling_price'>



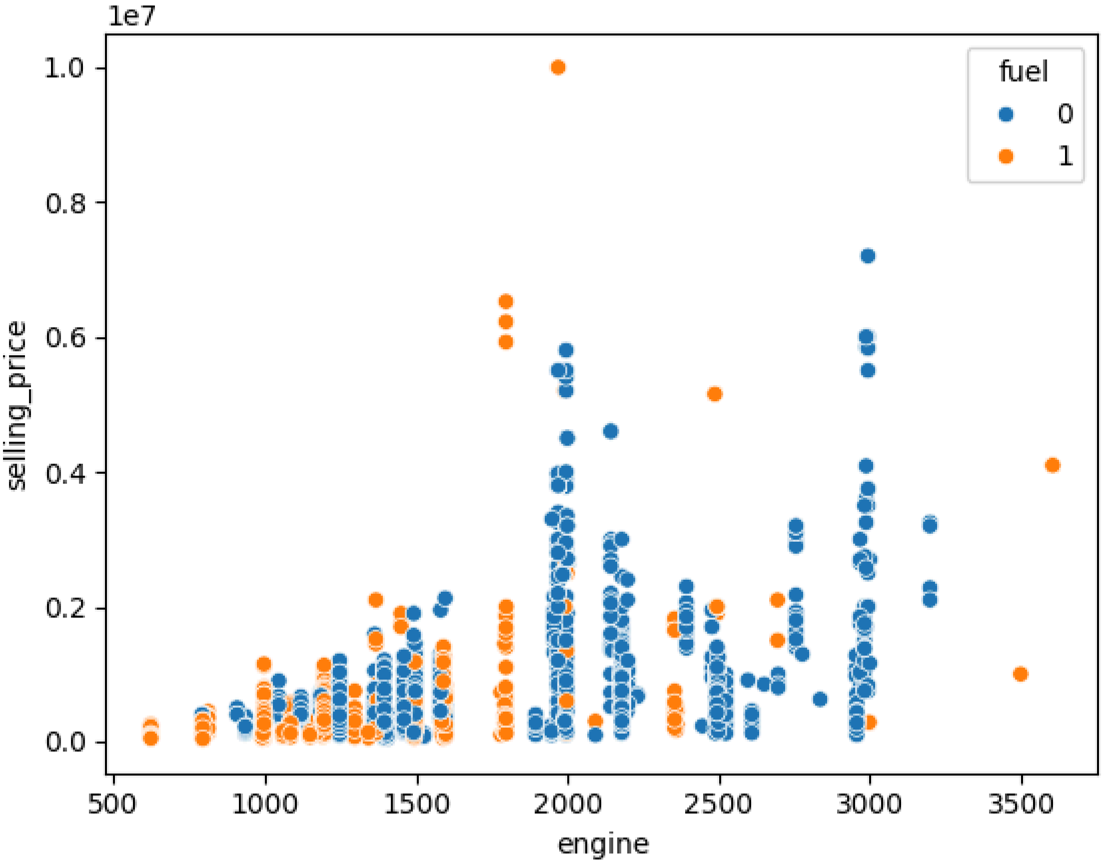
In [385…
```
sns.scatterplot(x=df['mileage'], y=df['selling_price'], hue=df['fuel'])
```

Out[385… <Axes: xlabel='mileage', ylabel='selling_price'>



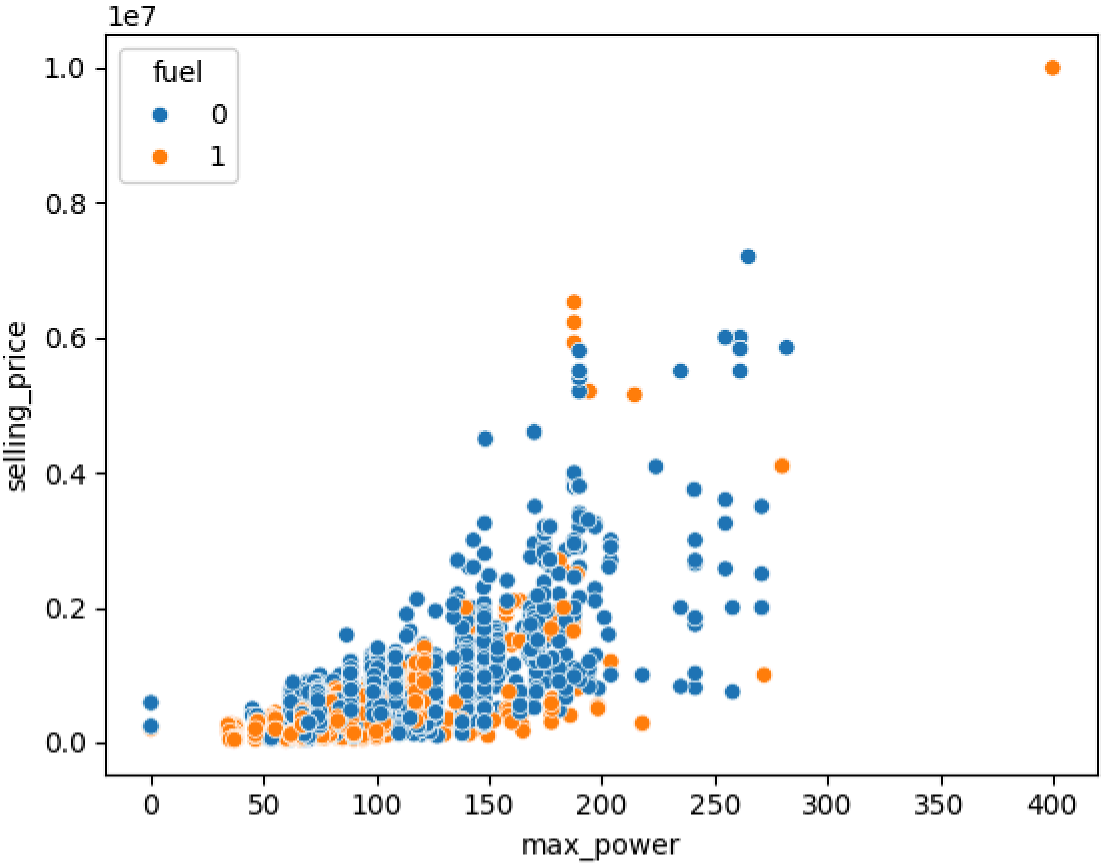
In [386…
```
sns.scatterplot(x=df['engine'], y=df['selling_price'], hue=df['fuel'])
```

Out[386… <Axes: xlabel='engine', ylabel='selling_price'>



In [387…
```
sns.scatterplot(x=df['max_power'], y=df['selling_price'], hue=df['fuel'])
```

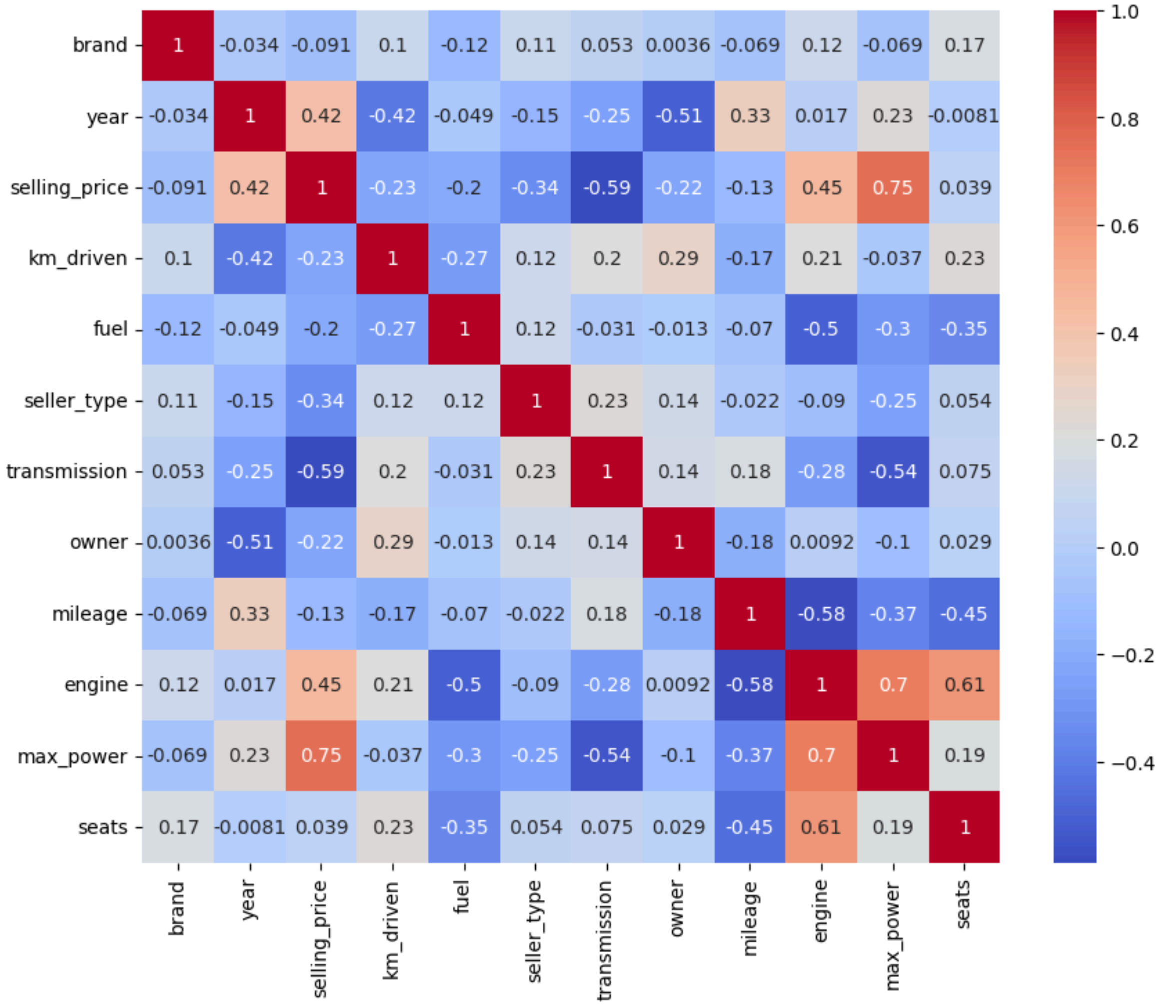
Out[387… <Axes: xlabel='max_power', ylabel='selling_price'>



Correlation matrix

In [388…

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Feature Selection

In [389…

```
#The important features that were selected based on the correlation and p

# Main features = max power, mileage, engine and brand

X = df[['max_power', 'mileage', 'year', 'brand']]

#Selling price is stored in variable Y, which is the target variable. Her
y = np.log(df['selling_price'])
```

Data Splitting

We are splitting the values into train set and test set.

In [390…

```
from sklearn.model_selection import train_test_split
#Splitting the dataset into training and testing sets using train_test_sp
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

Preprocessing

Checking for null values in X_train

In [391…
```
#Checking for null values in the training and testing sets
X_train[['max_power', 'mileage', 'year', 'brand']].isnull().sum()
```

Out[391…
```
max_power    146
mileage      151
year           0
brand          0
dtype: int64
```

In [392…
```
X_test[['max_power', 'mileage', 'year', 'brand']].isnull().sum()
```

Out[392…
```
max_power    62
mileage      63
year          0
brand         0
dtype: int64
```

In [393…
```
#Checking for null values in y_train
y_train.isnull().sum()
```

Out[393…
```
np.int64(0)
```

In [394…
```
#Checking for null values in y_train
y_test.isnull().sum()
```

Out[394…
```
np.int64(0)
```

Finding the mean, median, and mode of the features to fill up the null values

Here only max_power and mileage have null values among the features.

In [395…
```
X_train['max_power'].median() #Finding the median of max_power because th
```

Out[395…
```
np.float64(83.1)
```

In [396…
```
X_train['mileage'].mean() #Finding the mean of mileage because the distri
```

Out[396…
```
np.float64(19.35297697368421)
```

In [397…
```
X_test['max_power'].median()
```

Out[397…
```
np.float64(82.0)
```

In [398…
```
X_test['mileage'].mean()
```

Out[398…
```
np.float64(19.47756710694504)
```

Filling the missing numerical values

Using the obtained median and mean of test and train dataset to fill in the null values.

In [399…
```
X_train['max_power'].fillna(X_train['max_power'].median(), inplace=True)
X_train['mileage'].fillna(X_train['mileage'].mean(), inplace=True)
X_test['max_power'].fillna(X_test['max_power'].median(), inplace=True)
X_test['mileage'].fillna(X_test['mileage'].mean(), inplace=True)
```

In [400…
```
#Verifying if the null values are filled in training set
X_train[['max_power', 'mileage', 'brand', 'year']].isnull().sum()
```

Out[400…
```
max_power    0
mileage      0
brand        0
year         0
dtype: int64
```

In [401…
```
#Verifying if the null values are filled in test set
X_test[['max_power', 'mileage', 'brand', 'year']].isnull().sum()
```

Out[401…
```
max_power    0
mileage      0
brand        0
year         0
dtype: int64
```

In [402…
```
#Verifying if the null values are filled in y_train and y_test
y_train.isnull().sum(), y_test.isnull().sum()
```

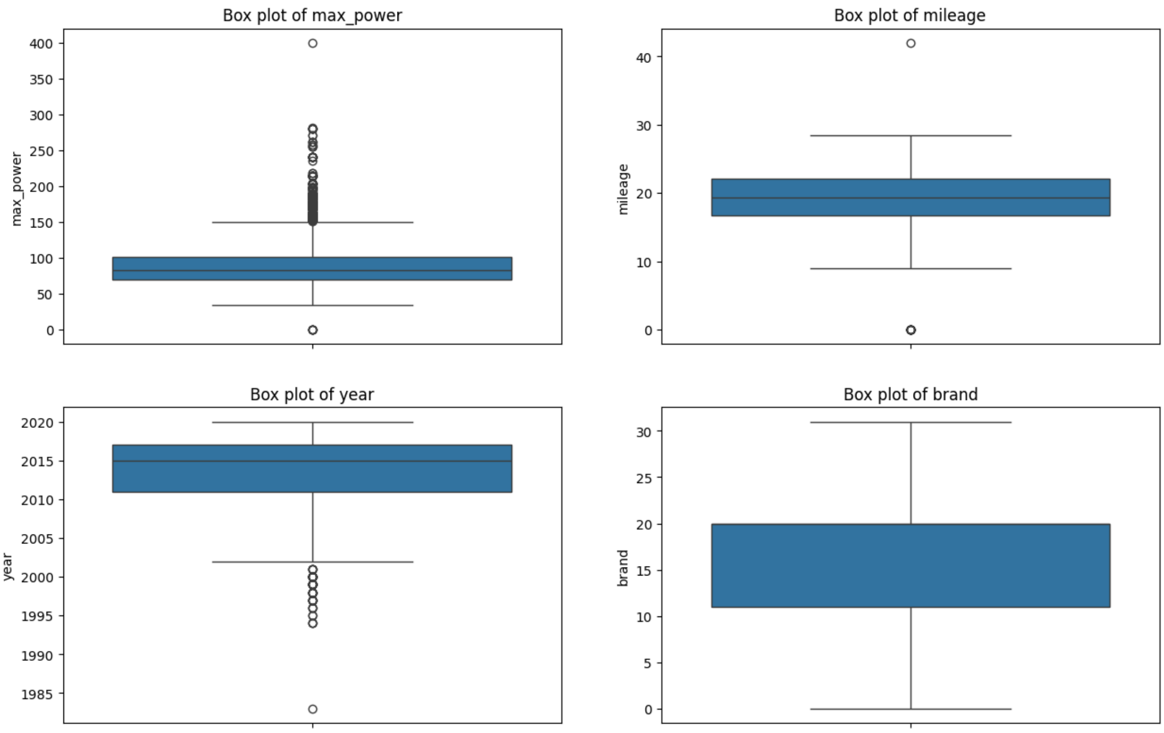
Out[402…
```
(np.int64(0), np.int64(0))
```

Checking for outliers

In [403…
```
#Checking the outliers in the training set using box plot
#Creating a dictionary to store the columns and their respective values

col_dict = {'max_power': 1, 'mileage': 2, 'year': 3, 'brand': 4}

#Using boxplot to visualize the outliers in the training set
plt.figure(figsize=(15,20))
for i, col in col_dict.items():
    plt.subplot(4,2,col)
    sns.boxplot(X_train[i])
    plt.title(f'Box plot of {i}')
plt.show()
```



In [404…

```
#To check the impact of outliers on the efficiency of the model, we will

def calculate_outliers(col, data=X_train):
    q75 = np.percentile(data[col], 75)
    q25 = np.percentile(data[col], 25)
    iqr = q75 - q25
    min_val = q25 - (1.5 * iqr)
    max_val = q75 + (1.5 * iqr)

    calculate_outliers = len(np.where((data[col] < min_val) | (data[col]
    percentage_outliers = round(calculate_outliers / len(data) * 100, 2)

    if calculate_outliers > 0:
        print(f'Feature {col} has {calculate_outliers} outliers which is
    else:
        print(f'Feature {col} has no outliers')
```

In [405…

```
for col in X_train.columns:
    calculate_outliers(col)
```

```
Feature max_power has 412 outliers which is 7.33% of the data
Feature mileage has 13 outliers which is 0.23% of the data
Feature year has 55 outliers which is 0.98% of the data
Feature brand has no outliers
```

Since the percentage of the outliers are low in the dataset, we are neglecting them for analysis

Scaling the dataset

In [406…

```
#Importing the library from sklearn for scaling the features, excluding t

from sklearn.preprocessing import MinMaxScaler
```

```
num_cols = ['max_power', 'mileage', 'year']
scaler = MinMaxScaler(feature_range=(0,1))
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

In [407…

```
#Verifying the changes made
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5623, 4)
(2410, 4)
(5623,)
(2410,)
```

Modelling

In [408…

```
#Using linear regression for the initial analysis
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_s
lr = LinearRegression()
lr.fit(X_train, y_train)
yhat = lr.predict(X_test)

print('Mean Absolute Error:', mean_absolute_error(y_test, yhat))
print('r2 score:', r2_score(y_test, yhat))
```

```
Mean Absolute Error: 0.2600134522445348
r2 score: 0.8368629406767405
```

Performing cross validation

In [409…

```
#Importing the libraries for performing cross validation
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor

algorithms = [LinearRegression(), SVR(), RandomForestRegressor(n_estimato

algorithm_names = ['Linear Regression', 'Support Vector Regression', 'Ran
```

In [410…

```
from sklearn.model_selection import KFold, cross_val_score

train_mse = []
test_mse = []

kfold = KFold(n_splits=5, shuffle=True)

for i, model in enumerate(algorithms):
```

```
    scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='
    print(f"{algorithm_names[i]} - Score: {scores}; Mean: {scores.mean()}
```

```
Linear Regression - Score: [-0.11447102 -0.12951146 -0.11793853 -0.1099763
4 -0.11580707]; Mean: -0.117540884264955 42
Support Vector Regression - Score: [-0.41625564 -0.46306308 -0.46292693 -
0.46464226 -0.40969417]; Mean: -0.4433164149771348
Random Forest Regressor - Score: [-0.05434064 -0.04905004 -0.05277671 -0.0
5365923 -0.05330087]; Mean: -0.052625499946119
Decision Tree Regressor - Score: [-0.05836573 -0.06663371 -0.07687545 -0.0
6484177 -0.07238391]; Mean: -0.06782011343320174
K-Neighbors Regressor - Score: [-0.0526415  -0.05491845 -0.05184057 -0.053
0656  -0.06747424]; Mean: -0.055988071671686325
```

Grid Search

According to cross validation, random forest regressor is the model with the highest efficiency.

In [411…

```
from sklearn.model_selection import GridSearchCV

param_grid = {'bootstrap': [True], 'max_depth': [5, 10, None],
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15]}

rf = RandomForestRegressor(random_state = 1)

grid = GridSearchCV(estimator = rf,
                    param_grid = param_grid,
                    cv = kfold,
                    n_jobs = -1,
                    return_train_score=True,
                    refit=True,
                    scoring='neg_mean_squared_error')

# Fit your grid_search
grid.fit(X_train, y_train);  #fit means start looping all the possible pa
```

```
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
/Projects/Python-for-Machine-Learning/.python/cpython-3.13.6-linux-x86_64-
gnu/lib/python3.13/multiprocessing/queues.py:120: UserWarning: pkg_resourc
es is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_r
esources.html. The pkg_resources package is slated for removal as early as
2025-11-30. Refrain from using this package or pin to Setuptools<81.
  return _ForkingPickler.loads(res)
```

In [412…
```
#Getting the best parameters and best negative mean squared error from th
best_params = grid.best_params_
best_score = -grid.best_score_          #A '-' sign is added to ignore th

print("Best Parameters: ", best_params)
print("Best Mean Squared Error: ", best_score)
```

```
Best Parameters:  {'bootstrap': True, 'max_depth': None, 'n_estimators': 1
5}
Best Mean Squared Error:  0.05396162141026741
```

Testing

In [413…
```
yhat = grid.predict(X_test) #Storing the predicted values in yhat variabl
#Calculating the mean absolute error and r2 score of the model

print('Mean Absolute Error:', mean_absolute_error(y_test, yhat))
print('r2 score:', r2_score(y_test, yhat))
```

```
Mean Absolute Error: 0.15647022381321024
r2 score: 0.9245670318148385
```

Analysis: Feature Importance

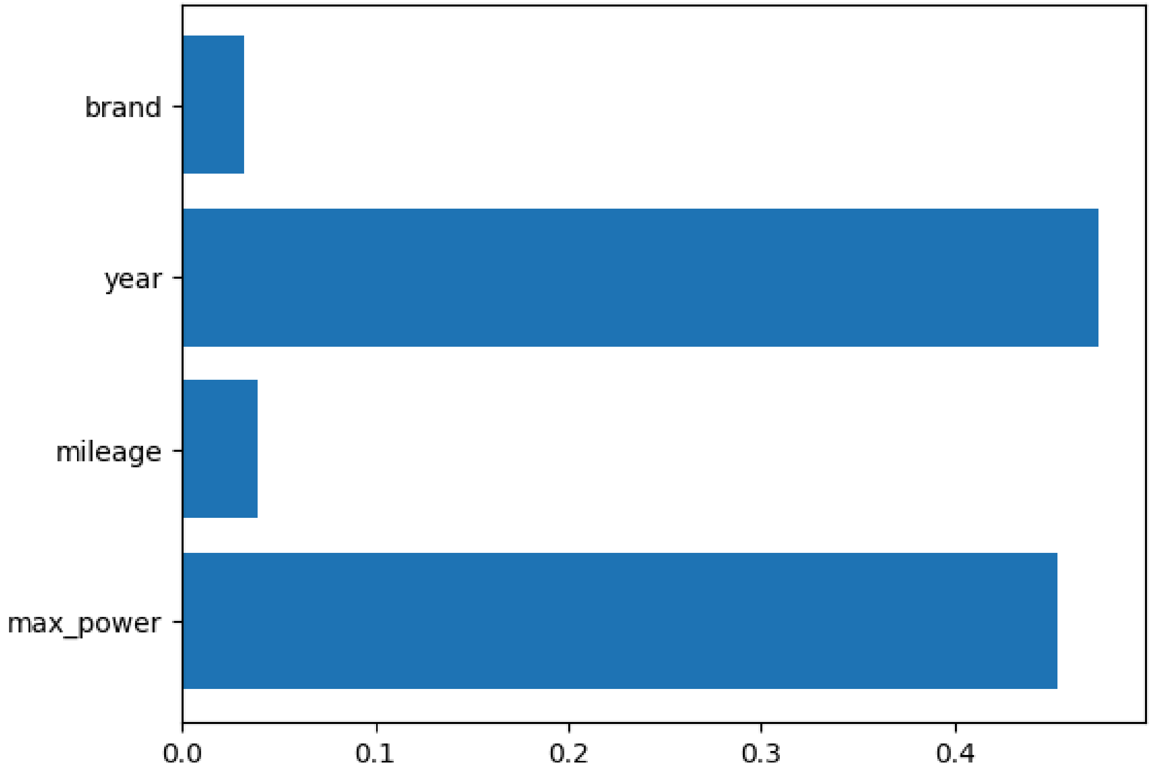
Analyzing the importance of each feature with the help of a bar graph

In [414…
```
rfr = grid.best_estimator_  #Storing the best estimator from the grid sea
rfr.feature_importances_   #Getting the feature importances from the mode
```

Out[414…
```
array([0.45354922, 0.03946125, 0.47516027, 0.03182926])
```

In [415…
```
#Plotting the bar graph of the feature importances
plt.barh(X.columns, rfr.feature_importances_)
```

Out[415…
```
<BarContainer object of 4 artists>
```



Inference

Checking the trained model with new and unseen data sets

In [416…
```
import pickle

# save the model to disk
filename = 'model/car_prediction.model'
pickle.dump (grid, open(filename, 'wb'))
label_encoder_brand_path = 'model/brand-label.model'
pickle.dump(label_encoder_brand, open(label_encoder_brand_path, 'wb'))

scaler_path = 'model/prediction_scalar.model'
pickle.dump(scaler, open(scaler_path, 'wb'))

feature_importance_path = 'model/feature_importance.model;'
pickle.dump(rfr, open(feature_importance_path, 'wb'))
```

In [417…
```
#Loading the model from the disk
loaded_model = pickle.load(open(filename, 'rb'))
scalar_model = pickle.load (open(scaler_path, 'rb'))
label_brand_model = pickle.load(open(label_encoder_brand_path, 'rb'))
feature_importances_model = pickle.load(open(feature_importance_path, 'rb
```

Working with an example

In [418…
```
sample = df[['max_power', 'mileage', 'year', 'brand']].loc[1].to_frame().
sample
```

Out[418…

	max_power	mileage	year	brand
1	103.52	21.14	2014.0	27.0

In [419…

```
sample [num_cols]
```

Out[419…

	max_power	mileage	year
1	103.52	21.14	2014.0

In [420…

```
sample [num_cols] = scalar_model.transform(sample[num_cols])
sample
```

Out[420…

	max_power	mileage	year	brand
1	0.2588	0.503333	0.837838	27.0

In [421…

```
predicted_selling_price = loaded_model.predict(sample)
print ("The predicted selling price is: ", str(np.exp(predicted_selling_p
```

```
The predicted selling price is:  [553113.39436598]
```

Report

The car prediction model is a basic machine learning model which predicts the price of a car based on the values of the features that the user has selected.

The initial data set contained the features like: name(brand), year, km_driven, fuel, setter_type, trainsmission, owner, mileage, engine, max_power, torque, and seats. For the analysis we dropped and cleaned the data first for the analysis. We separated the string from the numerical values for the features fuel, mileage, engine, and maximum power. After performing the initial cleaning, an explanatory data analysis was performed to understand the nature of the features and their interdependency. A univariate analysis was performed using a distribution plot to observe the distribution of the data sets. Scatter plot was used to see the relationship between features and selling price.

A correlation matrix was used to see the relation of the features with the selling price and select the most influential features. After analyzing the correlation matrix, maximum power, mileage, year, and brand were selected as important features for the analysis. After feature selection, the datasets were separated into train and test sets. About 30% of the dataset were separated into the test set. During the preprocessing step, median was used to fill the missing values of the maximum power and mean was used to fill the missing values of mileage. Our test set has no null values of the features that were selected.

Feature Selection

The features that were selected for analysis were as follows:

- Brand: The selling price of a car depends on the brand to some influential extent. The selling price of some luxury brands like BMW, Mercedes,etc. are higher than other brands.
- Year of manufacture: It influences the selling price of the car because newly manufactured cars are sold for higher selling price than old price.
- Maximum power: The cars with high power are priced higher due to their better performance and the use of higher level equipment.
- Mileage: Cars which provide good mileage and better fuel efficiency can increase the demand of the product, and hence create a rise in its price.

The other features like seats, seller type, and owner have lower influence than that of the selected features on the selling price.

Algorithm

Random forest regressor model was used for training the model as the model naturally handles non-linear relationships, feature interactions and performs well against outliers. The model also requires minimal preprocessing as compared to other models (for example, we did not need to use one hot encoding for ‘brand’ features).

The models which were not selected were:

- Linear regression: This model works the best when the relationship between the features is a straight line. Car prices don’t follow a straight line and features affect each other, so it misses important patterns.
- Single Decision Tree Easy to understand but tends to memorize the training data and make unsteady predictions. A Random Forest fixes this by averaging many trees.
- KNN Regression Predicts by averaging “nearby” examples. It’s sensitive to how data is scaled, slows down at prediction time, and struggles when data is less.

Hence, for a regression task like this the Random Forest Regressor has been selected.

Conclusions

Thus, the important features that were selected for the analysis were ‘max_power, ‘year’, ‘brand’, and ‘mileage’. Random Forest Regressor was used due to its better predictive performance.