Milestone 2

Model Building

1. What we want to predict is the "Price". We will use the normalized version 'price_log' for modeling.
2. Before we proceed to the model, we'll have to encode categorical features. We will drop categorical features like Name.
3. We'll split the data into train and test, to be able to evaluate the model that we build on the train data.
4. Build Regression models using train data.
5. Evaluate the model performance.

Note: Please load the data frame that was saved in Milestone 1 here before separating the data, and then proceed to the next step in Milestone 2.

Load the data

In [1]:

```
import pandas as pd

cars_data = pd.read_csv("cars_data_updated.csv")
```

Split the Data

- Step1: Seperating the indepdent variables (X) and the dependent variable (y).
- Step2: Encode the categorical variables in X using pd.dummies.
- Step3: Split the data into train and test using train_test_split.

Think about it: Why we should drop 'Name','Price','price_log','Kilometers_Driven' from X before splitting?

In [2]:

```
# Libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Regression models:
from sklearn.linear_model import LinearRegression  # For linear regression model
from sklearn.tree import DecisionTreeRegressor     # For decision tree model
from sklearn.linear_model import Ridge             # For Ridge / Lasso Regression
from sklearn.ensemble import RandomForestRegressor # For Random Forest regressor

# Metrics to evaluate the model
from sklearn.metrics import confusion_matrix, classification_report, f1_score
from sklearn.datasets import make_regression
from sklearn import metrics

from sklearn.model_selection import train_test_split
from sklearn import tree

# For hyperparameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
```

In [3]:

```
#familiarizing with dataset
cars_data.describe().T
```

Out[3]:

	count	mean	std	min	25%	50%	75%	max
Year	6018.0	2013.357594	3.269677	1998.000000	2011.000000	2014.000000	2016.000000	2019.000000
Kilometers_Driven	6018.0	57668.047690	37878.783175	171.000000	34000.000000	53000.000000	73000.000000	775000.000000
Mileage	6018.0	18.135329	4.581823	0.000000	15.170000	18.160000	21.100000	33.540000
Engine	6018.0	1618.510469	600.234304	72.000000	1197.000000	1493.000000	1969.000000	5998.000000
Power	6018.0	112.296265	53.537215	34.200000	74.000000	91.100000	138.030000	560.000000
Seats	6018.0	5.277999	0.803837	2.000000	5.000000	5.000000	5.000000	10.000000
Price	6018.0	9.470243	11.165926	0.440000	3.500000	5.640000	9.950000	160.000000
kilometers_driven_log	6018.0	10.757961	0.713022	5.141664	10.434116	10.878047	11.198215	13.560618
price_log	6018.0	1.824705	0.873606	-0.820981	1.252763	1.729884	2.297573	5.075174

In [4]:

```
#checking for null values
cars_data.isnull().sum()
```

Out[4]:

```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 0
Engine                  0
Power                   0
Seats                   0
Price                   0
Brand                   0
Model                   0
kilometers_driven_log   0
price_log               0
dtype: int64
```

In [3]:

```
cars_data.describe(include='object')
```

Out[3]:

	Name	Location	Fuel_Type	Transmission	Owner_Type	Brand	Model
count	6018	6018	6018	6018	6018	6018	6018
unique	1876	11	5	2	4	30	211
top	Mahindra XUV500 W8 2WD	Mumbai	Diesel	Manual	First	maruti	swift
freq	49	790	3204	4299	4928	1211	353

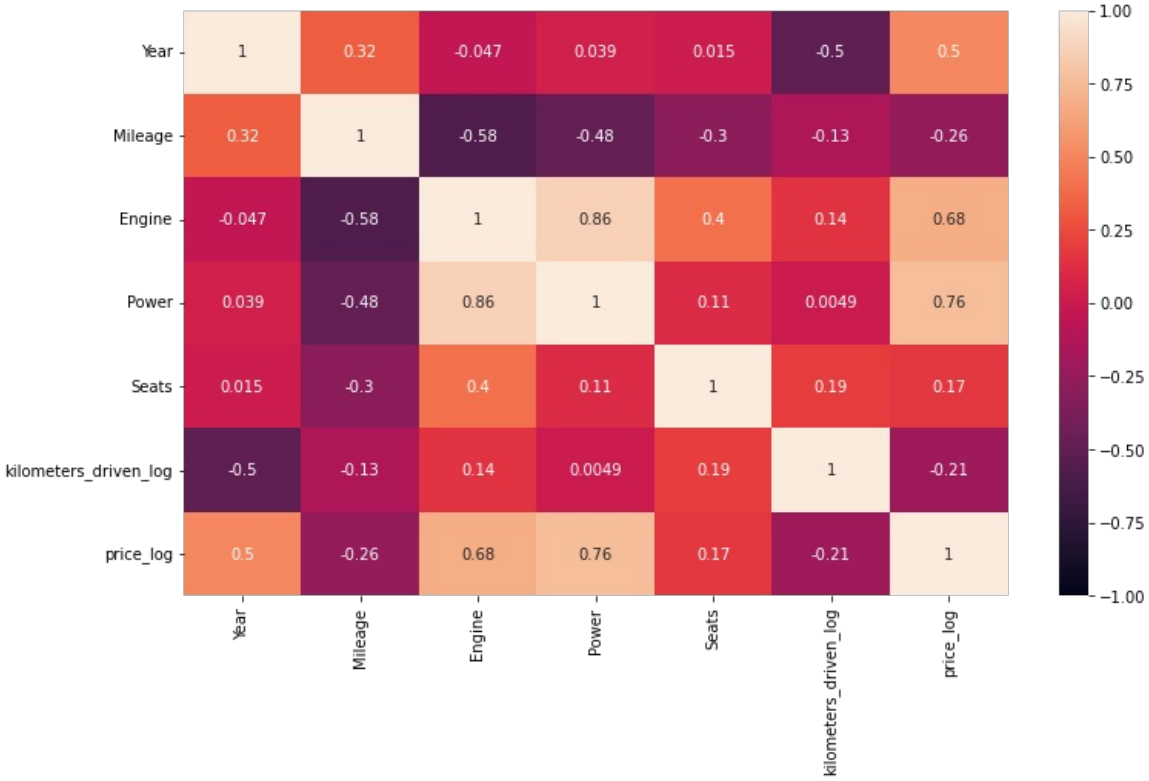
In [4]:

```
#I used a heatmap to understand the relationships and correlations between the variables.
#We notice that Power, Engine, Year have the highest correlation with our variable of interest “price_log.”

plt.figure(figsize = (12, 7))

sns.heatmap(cars_data.drop(['Price', 'Kilometers_Driven'],axis = 1).corr(), annot = True, vmin = -1, vmax = 1)

plt.show()
```



In [5]:

```
#looking at the mean price for each brand of cars
cars_data.groupby(["Brand"])["Price"].mean().sort_values(ascending=False)
```

Out[5]:

```
Brand
lamborghini      120.000000
bentley           59.000000
porsche           48.348333
land              39.259500
jaguar            37.632250
mini              26.896923
mercedes-benz     26.809874
audi              25.537712
bmw               25.093684
volvo             18.802857
jeep              18.718667
isuzu             14.696667
toyota            11.580024
mitsubishi        11.058889
force              9.333333
mahindra           8.045919
skoda              7.559075
ford               6.889400
renault            5.799034
honda              5.411743
hyundai            5.343433
volkswagen         5.307270
nissan             4.738352
maruti             4.517267
tata               3.562849
fiat               3.269286
datsun             3.049231
chevrolet          3.044463
smart              3.000000
ambassador         1.350000
Name: Price, dtype: float64
```

In [6]:

```
# Step-1
X = cars_data.drop(['Name','Price','price_log','Kilometers_Driven'], axis = 1)

y = cars_data[["price_log", "Price"]]
```

In [7]:

```
# Step-2 Use pd.get_dummies(drop_first = True)
X = pd.get_dummies(X, drop_first = True)
```

In [8]:

```
# Step-3 Splitting data into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

print(X_train.shape, X_test.shape)
```

```
(4212, 263) (1806, 263)
```

In [9]:

```
# Writing a function for calculating r2_score and RMSE on train and test data
# This function takes model as an input on which we have trained particular algorithm
# The categorical column as the input and returns the boxplots and histograms for the variable

def get_model_score(model, flag = True):
    '''
    model : regressor to predict values of X

    '''
    # Defining an empty list to store train and test results
    score_list = []

    pred_train = model.predict(X_train)

    pred_train_ = np.exp(pred_train)

    pred_test = model.predict(X_test)

    pred_test_ = np.exp(pred_test)

    train_r2 = metrics.r2_score(y_train['Price'], pred_train_)

    test_r2 = metrics.r2_score(y_test['Price'], pred_test_)

    train_rmse = metrics.mean_squared_error(y_train['Price'], pred_train_, squared = False)

    test_rmse = metrics.mean_squared_error(y_test['Price'], pred_test_, squared = False)

    # Adding all scores in the list
    score_list.extend((train_r2, test_r2, train_rmse, test_rmse))

    # If the flag is set to True then only the following print statements will be dispayed, the default value is
True
    if flag == True:

        print("R-sqaure on training set : ", metrics.r2_score(y_train['Price'], pred_train_))

        print("R-square on test set : ", metrics.r2_score(y_test['Price'], pred_test_))

        print("RMSE on training set : ", np.sqrt(metrics.mean_squared_error(y_train['Price'], pred_train_)))

        print("RMSE on test set : ", np.sqrt(metrics.mean_squared_error(y_test['Price'], pred_test_)))

    # Returning the list with train and test scores
    return score_list
```

For Regression Problems, some of the algorithms used are :

1) Linear Regression
2) Ridge / Lasso Regression
3) Decision Trees
4) Random Forest

Fitting a linear model

Linear Regression can be implemented using:

1) Sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
2) Statsmodels: https://www.statsmodels.org/stable/regression.html (https://www.statsmodels.org/stable/regression.html)

In [10]:

```
# Create a linear regression model
lr = LinearRegression()
```

In [11]:

```
# Fit linear regression model
lr.fit(X_train, y_train['price_log'])
```

Out[11]:

```
LinearRegression()
```

In [12]:

```
# Get score of the model
LR_score = get_model_score(lr)
```

```
R-sqaure on training set :  0.9405443096513909
R-square on test set :  0.8791796165653779
RMSE on training set :  2.724256408931264
RMSE on test set :  3.8739101364088144
```

Observations from results: _

- Our model has a high r-squared value, which is equal 0.94, but unfortunately r-squared for the test set is lower.
- This means that our model is overfitting

Important variables of Linear Regression

Building a model using statsmodels.

In [13]:

```
# Import Statsmodels
import statsmodels.api as sm

# Statsmodel api does not add a constant by default. We need to add it explicitly
x_train = sm.add_constant(X_train)

# Add constant to test data
x_test = sm.add_constant(X_test)

def build_ols_model(train):

    # Create the model
    olsmodel = sm.OLS(y_train["price_log"], train)

    return olsmodel.fit()


# Fit linear model on new dataset
olsmodel1 = build_ols_model(x_train)

print(olsmodel1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.959
Model:                            OLS   Adj. R-squared:                  0.956
Method:                 Least Squares   F-statistic:                     407.6
Date:                Thu, 06 Oct 2022   Prob (F-statistic):              0.00
Time:                        20:59:47   Log-Likelihood:                1317.6
No. Observations:                4212   AIC:                            -2179.
Df Residuals:                    3984   BIC:                            -732.4
Df Model:                         227
Covariance Type:            nonrobust
==============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
```

const	-204.4926	2.877	-71.069	0.000	-210.134	-198.851
Year	0.1073	0.001	72.710	0.000	0.104	0.110
Mileage	-0.0007	0.001	-0.452	0.651	-0.004	0.002
Engine	-8.661e-05	2.19e-05	-3.956	0.000	-0.000	-4.37e-05
Power	0.0024	0.000	10.163	0.000	0.002	0.003
Seats	0.0014	0.015	0.093	0.926	-0.027	0.030
kilometers_driven_log	-0.0736	0.006	-13.074	0.000	-0.085	-0.063
Location_Bangalore	0.1852	0.019	9.850	0.000	0.148	0.222
Location_Chennai	0.0593	0.018	3.305	0.001	0.024	0.094
Location_Coimbatore	0.1532	0.017	8.955	0.000	0.120	0.187
Location_Delhi	-0.0811	0.017	-4.686	0.000	-0.115	-0.047
Location_Hyderabad	0.1478	0.017	8.870	0.000	0.115	0.181
Location_Jaipur	-0.0193	0.018	-1.051	0.293	-0.055	0.017
Location_Kochi	-0.0173	0.017	-1.012	0.311	-0.051	0.016
Location_Kolkata	-0.2193	0.018	-12.509	0.000	-0.254	-0.185
Location_Mumbai	-0.0507	0.017	-3.049	0.002	-0.083	-0.018
Location_Pune	-0.0236	0.017	-1.379	0.168	-0.057	0.010
Fuel_Type_Diesel	0.0594	0.033	1.812	0.070	-0.005	0.124
Fuel_Type_Electric	-0.2914	0.088	-3.329	0.001	-0.463	-0.120
Fuel_Type_LPG	-0.0623	0.076	-0.815	0.415	-0.212	0.088
Fuel_Type_Petrol	-0.0596	0.034	-1.767	0.077	-0.126	0.007
Transmission_Manual	-0.1175	0.010	-11.475	0.000	-0.138	-0.097
Owner_Type_Fourth & Above	-0.1149	0.087	-1.321	0.187	-0.285	0.056
Owner_Type_Second	-0.0550	0.008	-6.531	0.000	-0.072	-0.039
Owner_Type_Third	-0.1347	0.022	-6.092	0.000	-0.178	-0.091
Brand_audi	-6.9817	0.112	-62.132	0.000	-7.202	-6.761
Brand_bentley	-3.3440	0.111	-30.054	0.000	-3.562	-3.126
Brand_bmw	-8.2702	0.158	-52.474	0.000	-8.579	-7.961
Brand_chevrolet	-8.3999	0.114	-73.937	0.000	-8.623	-8.177
Brand_datsun	-7.4411	0.109	-68.581	0.000	-7.654	-7.228
Brand_fiat	-8.1308	0.112	-72.449	0.000	-8.351	-7.911
Brand_force	-4.2364	0.091	-46.713	0.000	-4.414	-4.059
Brand_ford	-7.9583	0.113	-70.702	0.000	-8.179	-7.738
Brand_honda	-8.2506	0.115	-71.874	0.000	-8.476	-8.026
Brand_hyundai	-9.2602	0.169	-54.762	0.000	-9.592	-8.929
Brand_isuzu	-5.7181	0.114	-50.255	0.000	-5.941	-5.495
Brand_jaguar	-5.5570	0.105	-52.807	0.000	-5.763	-5.351
Brand_jeep	-4.2178	0.068	-61.642	0.000	-4.352	-4.084
Brand_lamborghini	-3.1921	0.114	-28.054	0.000	-3.415	-2.969
Brand_land	-3.7838	0.065	-58.654	0.000	-3.910	-3.657
Brand_mahindra	-8.4702	0.122	-69.621	0.000	-8.709	-8.232
Brand_maruti	-8.5905	0.134	-63.981	0.000	-8.854	-8.327
Brand_mercedes-benz	-7.3417	0.131	-55.971	0.000	-7.599	-7.085
Brand_mini	-5.9187	0.112	-52.988	0.000	-6.138	-5.700
Brand_mitsubishi	-6.7058	0.113	-59.296	0.000	-6.928	-6.484
Brand_nissan	-7.7128	0.114	-67.579	0.000	-7.937	-7.489
Brand_porsche	-5.4080	0.106	-51.018	0.000	-5.616	-5.200
Brand_renault	-8.1178	0.114	-71.386	0.000	-8.341	-7.895
Brand_skoda	-7.6054	0.105	-72.514	0.000	-7.811	-7.400
Brand_smart	-4.5206	0.111	-40.838	0.000	-4.738	-4.304
Brand_tata	-8.4757	0.147	-57.575	0.000	-8.764	-8.187
Brand_toyota	-7.2673	0.112	-64.728	0.000	-7.487	-7.047
Brand_volkswagen	-7.7775	0.112	-69.374	0.000	-7.997	-7.558
Brand_volvo	-6.7864	0.113	-59.864	0.000	-7.009	-6.564
Model_1000	3.732e-14	7.92e-16	47.145	0.000	3.58e-14	3.89e-14
Model_3	0.1140	0.108	1.056	0.291	-0.098	0.326
Model_5	0.4102	0.109	3.769	0.000	0.197	0.624
Model_6	0.8410	0.154	5.447	0.000	0.538	1.144
Model_7	0.8270	0.124	6.667	0.000	0.584	1.070
Model_800	-1.3935	0.093	-15.048	0.000	-1.575	-1.212
Model_a	-0.8510	0.107	-7.917	0.000	-1.062	-0.640
Model_a-star	-0.8159	0.086	-9.505	0.000	-0.984	-0.648
Model_a3	-1.2036	0.089	-13.554	0.000	-1.378	-1.029
Model_a4	-1.0992	0.040	-27.656	0.000	-1.177	-1.021
Model_a6	-0.9670	0.042	-22.958	0.000	-1.050	-0.884
Model_a7	1.189e-15	3.42e-16	3.473	0.001	5.18e-16	1.86e-15
Model_a8	-0.0395	0.166	-0.237	0.812	-0.365	0.287
Model_accent	-0.2929	0.135	-2.173	0.030	-0.557	-0.029
Model_accord	-0.5325	0.049	-10.802	0.000	-0.629	-0.436
Model_alto	-1.0819	0.064	-16.884	0.000	-1.207	-0.956
Model_amaze	-0.9987	0.033	-30.512	0.000	-1.063	-0.935
Model_ameo	-1.4888	0.056	-26.607	0.000	-1.599	-1.379
Model_aspire	-1.1623	0.090	-12.873	0.000	-1.339	-0.985
Model_aveo	-1.1790	0.061	-19.328	0.000	-1.299	-1.059
Model_avventura	-1.0351	0.102	-10.108	0.000	-1.236	-0.834
Model_b	-0.9313	0.102	-9.087	0.000	-1.132	-0.730
Model_baleno	-0.5785	0.067	-8.662	0.000	-0.709	-0.448
Model_beat	-1.2135	0.044	-27.308	0.000	-1.301	-1.126
Model_beetle	4.51e-15	4.88e-16	9.245	0.000	3.55e-15	5.47e-15
Model_bolero	-0.3795	0.065	-5.845	0.000	-0.507	-0.252
Model_bolt	-1.0671	0.142	-7.511	0.000	-1.346	-0.789
Model_boxster	-3.787e-15	3.38e-16	-11.188	0.000	-4.45e-15	-3.12e-15

Model_br-v	-0.7816	0.123	-6.378	0.000	-1.022	-0.541
Model_brio	-1.1058	0.038	-29.041	0.000	-1.180	-1.031
Model_brv	-0.6878	0.082	-8.405	0.000	-0.848	-0.527
Model_c-class	-0.7603	0.148	-5.135	0.000	-1.051	-0.470
Model_camry	-0.9435	0.079	-12.002	0.000	-1.098	-0.789
Model_captiva	-0.5965	0.167	-3.571	0.000	-0.924	-0.269
Model_captur	-0.7677	0.120	-6.409	0.000	-1.003	-0.533
Model_cayenne	-2.6481	0.083	-31.866	0.000	-2.811	-2.485
Model_cayman	-1.1845	0.150	-7.873	0.000	-1.479	-0.890
Model_cedia	-1.86e-15	4.03e-16	-4.611	0.000	-2.65e-15	-1.07e-15
Model_celerio	-0.8822	0.067	-13.129	0.000	-1.014	-0.750
Model_ciaz	-0.3698	0.066	-5.571	0.000	-0.500	-0.240
Model_city	-0.6884	0.027	-25.199	0.000	-0.742	-0.635
Model_civic	-0.7748	0.041	-19.070	0.000	-0.854	-0.695
Model_cla	-0.7003	0.087	-8.027	0.000	-0.871	-0.529
Model_classic	-9.0524	0.212	-42.796	0.000	-9.467	-8.638
Model_cls-class	-0.0963	0.192	-0.502	0.616	-0.473	0.280
Model_clubman	-1.8043	0.148	-12.180	0.000	-2.095	-1.514
Model_compass	-4.2178	0.068	-61.642	0.000	-4.352	-4.084
Model_continental	-3.3440	0.111	-30.054	0.000	-3.562	-3.126
Model_cooper	-1.8968	0.080	-23.822	0.000	-2.053	-1.741
Model_corolla	-1.4602	0.041	-35.304	0.000	-1.541	-1.379
Model_countryman	-2.2175	0.149	-14.924	0.000	-2.509	-1.926
Model_cr-v	-0.1665	0.044	-3.757	0.000	-0.253	-0.080
Model_creta	0.5832	0.126	4.629	0.000	0.336	0.830
Model_crosspolo	-1.4638	0.117	-12.514	0.000	-1.693	-1.234
Model_cruze	-0.6956	0.063	-11.042	0.000	-0.819	-0.572
Model_d-max	-3.1420	0.114	-27.566	0.000	-3.365	-2.919
Model_duster	-0.7989	0.043	-18.613	0.000	-0.883	-0.715
Model_dzire	-0.4854	0.082	-5.916	0.000	-0.646	-0.325
Model_e	-4.436e-16	1.47e-16	-3.013	0.003	-7.32e-16	-1.55e-16
Model_e-class	-0.5590	0.072	-7.766	0.000	-0.700	-0.418
Model_ecosport	-0.9952	0.043	-23.190	0.000	-1.079	-0.911
Model_eeco	-1.0110	0.084	-11.970	0.000	-1.177	-0.845
Model_elantra	0.6000	0.132	4.544	0.000	0.341	0.859
Model_elite	0.1924	0.133	1.448	0.148	-0.068	0.453
Model_endeavour	-0.3109	0.061	-5.133	0.000	-0.430	-0.192
Model_enjoy	-0.9577	0.074	-12.966	0.000	-1.103	-0.813
Model_eon	-0.4010	0.125	-3.218	0.001	-0.645	-0.157
Model_ertiga	-0.2911	0.074	-3.918	0.000	-0.437	-0.145
Model_esteem	-1.0923	0.102	-10.760	0.000	-1.291	-0.893
Model_estilo	-0.7709	0.121	-6.378	0.000	-1.008	-0.534
Model_etios	-1.9120	0.045	-42.717	0.000	-2.000	-1.824
Model_evalia	-1.6382	0.163	-10.055	0.000	-1.958	-1.319
Model_f	-1.6305	0.148	-11.018	0.000	-1.921	-1.340
Model_fabia	-1.8006	0.060	-30.032	0.000	-1.918	-1.683
Model_fiesta	-1.3275	0.049	-26.979	0.000	-1.424	-1.231
Model_figo	-1.4450	0.043	-33.633	0.000	-1.529	-1.361
Model_fluence	-1.0426	0.100	-10.467	0.000	-1.238	-0.847
Model_fortuner	-0.7012	0.042	-16.889	0.000	-0.783	-0.620
Model_fortwo	-4.5206	0.111	-40.838	0.000	-4.738	-4.304
Model_freestyle	-0.4038	0.167	-2.412	0.016	-0.732	-0.076
Model_fusion	-1.0863	0.167	-6.500	0.000	-1.414	-0.759
Model_gallardo	-3.1921	0.114	-28.054	0.000	-3.415	-2.969
Model_getz	-0.2869	0.137	-2.101	0.036	-0.555	-0.019
Model_gl-class	0.0384	0.096	0.399	0.690	-0.150	0.227
Model_gla	-0.6050	0.090	-6.701	0.000	-0.782	-0.428
Model_glc	-0.2867	0.096	-2.975	0.003	-0.476	-0.098
Model_gle	-0.1329	0.088	-1.503	0.133	-0.306	0.040
Model_gls	-0.0214	0.195	-0.110	0.913	-0.404	0.362
Model_go	-2.2924	0.100	-22.882	0.000	-2.489	-2.096
Model_grand	-0.0796	0.121	-0.656	0.512	-0.317	0.158
Model_grande	-1.3411	0.102	-13.141	0.000	-1.541	-1.141
Model_hexa	-0.1350	0.201	-0.673	0.501	-0.528	0.258
Model_i10	-0.0687	0.123	-0.558	0.577	-0.310	0.172
Model_i20	0.1220	0.123	0.991	0.322	-0.119	0.363
Model_ignis	-0.9104	0.121	-7.551	0.000	-1.147	-0.674
Model_ikon	-1.5612	0.064	-24.580	0.000	-1.686	-1.437
Model_indica	-1.4149	0.102	-13.912	0.000	-1.614	-1.215
Model_indigo	-1.3190	0.104	-12.668	0.000	-1.523	-1.115
Model_innova	-1.0378	0.041	-25.343	0.000	-1.118	-0.958
Model_jazz	-0.9081	0.037	-24.736	0.000	-0.980	-0.836
Model_jeep	-0.1659	0.125	-1.322	0.186	-0.412	0.080
Model_jetta	-0.8736	0.051	-17.043	0.000	-0.974	-0.773
Model_koleos	-0.4870	0.101	-4.825	0.000	-0.685	-0.289
Model_kuv	-1.0008	0.082	-12.186	0.000	-1.162	-0.840
Model_kwid	-1.6528	0.049	-33.747	0.000	-1.749	-1.557
Model_lancer	-1.9803	0.119	-16.611	0.000	-2.214	-1.747
Model_laura	-1.2912	0.040	-32.206	0.000	-1.370	-1.213
Model_linea	-1.2215	0.074	-16.459	0.000	-1.367	-1.076
Model_lodgy	-0.8970	0.169	-5.295	0.000	-1.229	-0.565
Model_logan	-1.1515	0.125	-9.180	0.000	-1.397	-0.906
Model_m-class	-0.3060	0.084	-3.624	0.000	-0.472	-0.140

Model_manza	-1.1228	0.110	-10.181	0.000	-1.339	-0.907
Model_micra	-1.6362	0.057	-28.706	0.000	-1.748	-1.524
Model_mobilio	-0.8138	0.062	-13.074	0.000	-0.936	-0.692
Model_montero	-1.3789	0.157	-8.799	0.000	-1.686	-1.072
Model_mustang	0.3339	0.175	1.904	0.057	-0.010	0.678
Model_mux	-2.5761	0.137	-18.756	0.000	-2.845	-2.307
Model_nano	-1.8111	0.113	-15.959	0.000	-2.034	-1.589
Model_new	-0.6983	0.071	-9.895	0.000	-0.837	-0.560
Model_nexon	-0.4921	0.204	-2.415	0.016	-0.892	-0.093
Model_nuvosport	-0.8409	0.125	-6.750	0.000	-1.085	-0.597
Model_octavia	-1.1218	0.040	-28.206	0.000	-1.200	-1.044
Model_omni	-1.2249	0.079	-15.441	0.000	-1.380	-1.069
Model_one	-4.2364	0.091	-46.713	0.000	-4.414	-4.059
Model_optra	-0.9321	0.059	-15.876	0.000	-1.047	-0.817
Model_outlander	-1.8723	0.155	-12.054	0.000	-2.177	-1.568
Model_pajero	-1.4743	0.078	-18.934	0.000	-1.627	-1.322
Model_panamera	-1.5753	0.086	-18.299	0.000	-1.744	-1.407
Model_passat	-0.9526	0.079	-12.101	0.000	-1.107	-0.798
Model_petra	-1.5572	0.161	-9.651	0.000	-1.874	-1.241
Model_platinum	4.593e-16	6.57e-17	6.996	0.000	3.31e-16	5.88e-16
Model_polo	-1.4368	0.039	-37.276	0.000	-1.512	-1.361
Model_prius	-0.2914	0.088	-3.329	0.001	-0.463	-0.120
Model_pulse	-1.2715	0.089	-14.321	0.000	-1.446	-1.097
Model_punto	-1.4376	0.102	-14.110	0.000	-1.637	-1.238
Model_q3	-1.0742	0.051	-20.886	0.000	-1.175	-0.973
Model_q5	-0.7710	0.050	-15.566	0.000	-0.868	-0.674
Model_q7	-0.5774	0.056	-10.313	0.000	-0.687	-0.468
Model_qualis	-0.9212	0.128	-7.178	0.000	-1.173	-0.670
Model_quanto	-0.9193	0.090	-10.180	0.000	-1.096	-0.742
Model_r-class	-0.3853	0.127	-3.025	0.003	-0.635	-0.136
Model_rapid	-1.4712	0.038	-38.608	0.000	-1.546	-1.397
Model_redi	-2.6573	0.144	-18.486	0.000	-2.939	-2.376
Model_redi-go	-2.4914	0.086	-28.804	0.000	-2.661	-2.322
Model_renault	-0.7419	0.174	-4.275	0.000	-1.082	-0.402
Model_ritz	-0.7438	0.067	-11.114	0.000	-0.875	-0.613
Model_rover	-3.7838	0.065	-58.654	0.000	-3.910	-3.657
Model_rs5	-0.7455	0.127	-5.850	0.000	-0.995	-0.496
Model_s	-0.3076	0.064	-4.831	0.000	-0.432	-0.183
Model_s-class	-0.2563	0.112	-2.298	0.022	-0.475	-0.038
Model_s-cross	-0.4491	0.190	-2.365	0.018	-0.821	-0.077
Model_s60	-1.3777	0.081	-16.913	0.000	-1.537	-1.218
Model_s80	-1.8661	0.158	-11.809	0.000	-2.176	-1.556
Model_safari	-0.4804	0.120	-4.008	0.000	-0.715	-0.245
Model_sail	-1.0141	0.070	-14.408	0.000	-1.152	-0.876
Model_santa	0.7653	0.142	5.373	0.000	0.486	1.044
Model_santro	-0.2104	0.126	-1.676	0.094	-0.457	0.036
Model_scala	-1.2002	0.088	-13.600	0.000	-1.373	-1.027
Model_scorpio	-0.2345	0.043	-5.417	0.000	-0.319	-0.150
Model_siena	-1.5382	0.163	-9.428	0.000	-1.858	-1.218
Model_sl-class	0.2206	0.199	1.107	0.268	-0.170	0.611
Model_slc	-0.2771	0.151	-1.834	0.067	-0.573	0.019
Model_slk-class	-0.0087	0.131	-0.067	0.947	-0.266	0.249
Model_sonata	0.6582	0.150	4.380	0.000	0.364	0.953
Model_spark	-1.3119	0.074	-17.619	0.000	-1.458	-1.166
Model_ssangyong	-0.0191	0.064	-0.300	0.764	-0.144	0.106
Model_sumo	-0.5899	0.122	-4.846	0.000	-0.829	-0.351
Model_sunny	-1.4499	0.057	-25.337	0.000	-1.562	-1.338
Model_superb	-0.9498	0.036	-26.312	0.000	-1.021	-0.879
Model_swift	-0.5576	0.062	-8.965	0.000	-0.680	-0.436
Model_sx4	-0.5296	0.070	-7.561	0.000	-0.667	-0.392
Model_tavera	-0.4993	0.134	-3.725	0.000	-0.762	-0.236
Model_teana	-1.0449	0.162	-6.451	0.000	-1.362	-0.727
Model_terrano	-1.2318	0.057	-21.502	0.000	-1.344	-1.119
Model_thar	-0.4494	0.083	-5.445	0.000	-0.611	-0.288
Model_tiago	-1.1297	0.117	-9.629	0.000	-1.360	-0.900
Model_tigor	-0.9786	0.133	-7.354	0.000	-1.239	-0.718
Model_tiguan	-0.2744	0.162	-1.693	0.091	-0.592	0.043
Model_tt	-0.5044	0.122	-4.123	0.000	-0.744	-0.265
Model_tucson	0.7037	0.165	4.262	0.000	0.380	1.027
Model_tuv	-0.6730	0.071	-9.445	0.000	-0.813	-0.533
Model_v40	-1.2879	0.102	-12.618	0.000	-1.488	-1.088
Model_vento	-1.2875	0.039	-33.227	0.000	-1.363	-1.212
Model_venture	-0.9940	0.202	-4.909	0.000	-1.391	-0.597
Model_verito	-0.9115	0.104	-8.754	0.000	-1.116	-0.707
Model_verna	0.2536	0.125	2.035	0.042	0.009	0.498
Model_versa	-0.5424	0.196	-2.769	0.006	-0.926	-0.158
Model_vitara	-0.3587	0.070	-5.110	0.000	-0.496	-0.221
Model_wagon	-0.8330	0.064	-13.041	0.000	-0.958	-0.708
Model_wr-v	0	0	nan	nan	0	0
Model_wrv	-0.7927	0.168	-4.705	0.000	-1.123	-0.462
Model_x-trail	-0.7118	0.118	-6.010	0.000	-0.944	-0.480
Model_x1	0.0972	0.112	0.865	0.387	-0.123	0.318
Model_x3	0.4413	0.119	3.698	0.000	0.207	0.675

```
Model_x5                    0.7188      0.117      6.153      0.000       0.490       0.948
Model_x6                    0.9454      0.136      6.960      0.000       0.679       1.212
Model_xc60                 -1.3082      0.086    -15.203      0.000      -1.477      -1.139
Model_xc90                 -0.9466      0.161     -5.880      0.000      -1.262      -0.631
Model_xcent                -0.0647      0.126     -0.513      0.608      -0.312       0.183
Model_xe                         0          0        nan        nan           0           0
Model_xenon                -0.8828      0.139     -6.342      0.000      -1.156      -0.610
Model_xf                   -2.2456      0.068    -32.886      0.000      -2.379      -2.112
Model_xj                   -1.6809      0.083    -20.132      0.000      -1.845      -1.517
Model_xuv300               -0.1697      0.174     -0.977      0.329      -0.510       0.171
Model_xuv500               -0.1154      0.036     -3.202      0.001      -0.186      -0.045
Model_xylo                 -0.6978      0.058    -11.970      0.000      -0.812      -0.583
Model_yeti                 -0.9708      0.068    -14.250      0.000      -1.104      -0.837
Model_z4                    0.8746      0.217      4.032      0.000       0.449       1.300
Model_zen                  -0.9390      0.074    -12.755      0.000      -1.083      -0.795
Model_zest                 -0.9395      0.107     -8.741      0.000      -1.150      -0.729
==============================================================================
Omnibus:                     1982.883   Durbin-Watson:                   2.019
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           120844.480
Skew:                          -1.433   Prob(JB):                         0.00
Kurtosis:                      29.084   Cond. No.                     7.63e+19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.95e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [14]:

```
# Retrive Coeff values, p-values and store them in the dataframe
olsmod = pd.DataFrame(olsmodel1.params, columns = ['coef'])

olsmod['pval'] = olsmodel1.pvalues
```

In [15]:

```
# Filter by significant p-value (pval <= 0.05) and sort descending by Odds ratio

olsmod = olsmod.sort_values(by = "pval", ascending = False)

pval_filter = olsmod['pval']<= 0.05

olsmod[pval_filter]
```

Out[15]:

	coef	pval
Model_verna	0.253587	0.041950
Model_getz	-0.286947	0.035724
Model_accent	-0.292885	0.029862
Model_s-class	-0.256333	0.021635
Model_s-cross	-0.449056	0.018093
...	...	...
Brand_porsche	-5.408007	0.000000
Brand_nissan	-7.712777	0.000000
Brand_mitsubishi	-6.705807	0.000000
Brand_mini	-5.918719	0.000000
const	-204.492597	0.000000

231 rows × 2 columns

In [16]:

```
# We are looking are overall significant varaible

pval_filter = olsmod['pval']<= 0.05
mp_vars = olsmod[pval_filter].index.tolist()

# We are going to get overall varaibles (un-one-hot encoded varables) from categorical varaibles
sig_var = []
for col in mp_vars:
    if '' in col:
        first_part = col.split('_')[0]
        for c in cars_data.columns:
            if first_part in c and c not in sig_var :
                sig_var.append(c)


start = '\033[1m'
end = '\033[95m'
print(start+ 'Most overall significant categorical varaibles of LINEAR REGRESSION  are ' +end,':\n', sig_var)
```

```
Most overall significant categorical varaibles of LINEAR REGRESSION  are  :
 ['Model', 'Location', 'Fuel_Type', 'Engine', 'Owner_Type', 'Power', 'Transmission', 'kilometers_dri
ven_log', 'Brand', 'Year']
```

In [17]:

```
from sklearn.model_selection import cross_val_score

# Build the regression model and cross-validate
linearregression = LinearRegression()

cv_Score11 = cross_val_score(linearregression, X_train, y_train, cv = 10)
cv_Score12 = cross_val_score(linearregression, X_train, y_train, cv = 10,
                             scoring = 'neg_mean_squared_error')


print("RSquared: %0.3f (+/- %0.3f)" % (cv_Score11.mean(), cv_Score11.std() * 2))
print("Mean Squared Error: %0.3f (+/- %0.3f)" % (-1*cv_Score12.mean(), cv_Score12.std() * 2))
```

```
RSquared: 0.885 (+/- 0.055)
Mean Squared Error: 11.342 (+/- 10.729)
```

Build Ridge / Lasso Regression similar to Linear Regression:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

In [18]:

```
# Ridge regression model
rdg=Ridge()
rdg.fit(X_train, y_train['price_log'])
```

Out[18]:

```
Ridge()
```

In [19]:

```
# Score of the Ridge regression model
Ridge_score = get_model_score(rdg)
```

```
R-sqaure on training set :  0.9305904843647886
R-square on test set :  0.8974315477142721
RMSE on training set :  2.943477870528032
RMSE on test set :  3.569327009947088
```

Observations from results: _

- On test data Ridge regression model performs better than Linear Regression model
- However, this model is also overfitting considering that it performs better on training set than on test set

Decision Tree

https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html (https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html)

In [20]:

```
# Create a decision tree regression model, use random_state = 1
dtree = DecisionTreeRegressor(random_state=1)
```

In [21]:

```
# Fit decision tree regression model
dtree.fit(X_train, y_train['price_log'])
```

Out[21]:

```
DecisionTreeRegressor(random_state=1)
```

In [22]:

```
# Get score of the model
Dtree_model = get_model_score(dtree)
```

```
R-sqaure on training set :  0.9999965696959587
R-square on test set :  0.8296425759692035
RMSE on training set :  0.020692719736775493
RMSE on test set :  4.600022643909701
```

Observations from results: _

- This model performs extremely well on training data, but there's a much bigger gap with performance on test data
- This pattern of performance is not unusual for decision tree models

Print the importance of features in the tree building. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.

In [23]:

```
print(pd.DataFrame(dtree.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = 'Imp'
, ascending = False))
```

```
                          Imp
Power                0.613968
Year                 0.234875
Engine               0.036317
kilometers_driven_log  0.015771
Mileage              0.012068
...                       ...
Model_redi           0.000000
Model_r-class        0.000000
Model_qualis         0.000000
Model_classic        0.000000
Model_fortwo         0.000000

[263 rows x 1 columns]
```

Observations and insights: _

- The variables with highest impact on the output are listed in descending order.
- We notice that Power, Year and Engine have the highest coefficients.

Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

In [24]:

```
# Create a Randomforest regression model
rf = RandomForestRegressor(random_state = 1)
```

In [25]:

```
# Fit Randomforest regression model
rf.fit(X_train, y_train['price_log'])
```

Out[25]:

```
RandomForestRegressor(random_state=1)
```

In [26]:

```
# Get score of the model
RandomForest_model = get_model_score(rf)
```

```
R-sqaure on training set :  0.9781647231120805
R-square on test set :  0.8800929811425535
RMSE on training set :  1.6509374527551908
RMSE on test set :  3.8592395788134874
```

Observations and insights: _

- Random forest model is also overfitting

Feature Importance

In [28]:

```
# Print important features similar to decision trees
print(pd.DataFrame(rf.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = 'Imp', a
scending = False))
```

```
                           Imp
Power                 0.611794
Year                  0.232595
Engine                0.035978
kilometers_driven_log 0.018257
Mileage               0.014337
...                        ...
Model_cedia           0.000000
Model_1000            0.000000
Model_boxster         0.000000
Model_beetle          0.000000
Model_e               0.000000

[263 rows x 1 columns]
```

Observations and insights:

- We see that Power, Year, Engine, kilometers_driven_log have made the biggest impact on the outcomes of the RF model

Hyperparameter Tuning: Decision Tree

In [29]:

```
# The type of estimator
dtree_tuned = DecisionTreeRegressor(random_state = 1)

# Grid of parameters to choose from

parameters = {'max_depth': [None],
                'min_samples_leaf': [1, 3, 5, 7],
                 'max_leaf_nodes': [2, 5, 7] + [None],
            }

# Scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Running the grid search
grid_obj = GridSearchCV(dtree_tuned, parameters, scoring = scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train['price_log'])

# Set the model to the best combination of parameters
dtree_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data
dtree_tuned.fit(X_train, y_train['price_log'])
```

Out[29]:

```
DecisionTreeRegressor(min_samples_leaf=5, random_state=1)
```

In [31]:

```
# The score of the dtree_tuned
dtree_tuned_score = get_model_score(dtree_tuned)
```

```
R-sqaure on training set :  0.9204735568853475
R-square on test set :  0.803928517531618
RMSE on training set :  3.150699960846281
RMSE on test set :  4.934994278819966
```

Observations and insights:

- I tried different parameters to improve the performance of the model.
- However, hyperparameter tuning did not resolve the overfitting problem

Hyperparameter Tuning: Random Forest

In [ ]:

```
rf_tuned = RandomForestRegressor(random_state = 1)
# Define the parameters for Grid to choose from
parameters = {
                'max_depth': [5, 7, None],
                'max_features': ['sqrt', 'log2'],
                'n_estimators': [250, 500, 800, 900]
            }

scorer = metrics.make_scorer(metrics.r2_score)

grid_obj = GridSearchCV(rf_tuned, parameters, scoring = scorer, cv =5)
grib_obj = grid_obj.fit(X_train, y_train['price_log'])


rf_tuned = grid_obj.best_estimator_

rf_tuned.fit(X_train, y_train['price_log'])
```

In [ ]:

```
# Score of the model
rf_tuned_score = get_model_score(rf_tuned)
```

Observations and insights: _

- The performance of the model on test data slightly improved, but it is not significant

In [ ]:

```
# Defining list of models you have trained
models = [lr, rdg, dtree, dtree_tuned, rf, rf_tuned]

# Defining empty lists to add train and test results
r2_train = []
r2_test = []
rmse_train = []
rmse_test = []

# Looping through all the models to get the rmse and r2 scores
for model in models:

    # Accuracy score
    j = get_model_score(model, False)

    r2_train.append(j[0])

    r2_test.append(j[1])

    rmse_train.append(j[2])

    rmse_test.append(j[3])
```

In [47]:

```
comparison_frame = pd.DataFrame({'Model':['Linear Regression', 'Ridge Regression','Decision Tree', 'Random Forest
',
                                          'Tuned Decision Tree', 'Tuned Random Forest'],
                                          'Train_r2': r2_train,'Test_r2': r2_test,
                                          'Train_RMSE': rmse_train,'Test_RMSE': rmse_test})
comparison_frame
```

Out[47]:

	Model	Train_r2	Test_r2	Train_RMSE	Test_RMSE
0	Linear Regression	0.940544	0.879180	2.724256	3.873910
1	Ridge Regression	0.930590	0.897432	2.943478	3.569327
2	Decision Tree	0.999997	0.829643	0.020693	4.600023
3	Random Forest	0.920474	0.803929	3.150700	4.934994
4	Tuned Decision Tree	0.978165	0.880093	1.650937	3.859240
5	Tuned Random Forest	0.969545	0.855819	1.949761	4.231878

Observations:

- We see that Decision Tree has the highest r-squared value for training data, but its much lower for test data. Decision Trees usually tend to overfit.
- I recommend using the ridge regression model for finding the optimal price. It is an efficient formula, which can predict the best price match with fairly high accuracy.

In [ ]:

```
```