Used_Cars_Price_Prediction

May 22, 2025

1 Used Cars Price Prediction

Author: Sepehr Safari

1.1 Problem Definition

1.1.1 The Context:

Used car sales is one of the biggest market in India. Evey year handred of thousend of used cars are traded in this market. A good insight about how used car prices are influenced by factors like brand, mileage, power, engine and other specification of cars models which are playing key roles in determining and setting prices.

1.1.2 The objective:

As a Data Scientist, we are going to analyze the data, find out what factors affect the used car sales prices, and come up with a machine learning model which can predict the used car prices using the historical data available from car dealers in different location in India. Also, bring about useful insights and facts from the data, which can help to predict the used car prices based on different factores provided by provided data.

1.1.3 The key questions:

- What are the features or factors determining the used car prices.
- Which feature has the most positive influence on the price.
- What are the features or factors that have the most negative impact on the prices.

1.1.4 The problem formulation:

Estimating the price of used cars by taking into account a set of features, based on historical data. And then getting a better understanding on the most relevant features that help determine the price of an used vehicle. And at the end, find the right Regression model that can predict the most accurate price of an used car based on factors influencing its price in India market.

1.1.5 Data Dictionary

S.No. : Serial Number

Name : Name of the car which includes Brand name and Model name

Location : The location in which the car is being sold or is available for purchase (Cities)

Year : Manufacturing year of the car

Kilometers_driven : The total kilometers driven in the car by the previous owner(s) in KM

Fuel_Type : The type of fuel used by the car (Petrol, Diesel, Electric, CNG, LPG)

Transmission : The type of transmission used by the car (Automatic / Manual)

Owner : Type of ownership

Mileage : The standard mileage offered by the car company in kmpl or km/kg

Engine : The displacement volume of the engine in CC

Power : The maximum power of the engine in bhp

Seats : The number of seats in the car

New_Price : The price of a new car of the same model in INR 100,000

Price : The price of the used car in INR 100,000 (**Target Variable**)

1.1.6 Loading libraries

[1]:
```python
# Libraries for data exploration
import pandas as pd
import numpy as np

# Libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Add libraries for linear regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# To be used for data scaling and one hot encoding
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder

# To impute missing values
from sklearn.impute import SimpleImputer

# To do hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# To be used for creating pipelines and personalizing them
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Add StateModels libraries
from statsmodels.formula.api import ols
```

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_white
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeRegressor, plot_tree, export_text
from sklearn.ensemble import RandomForestRegressor

# To suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Library for uploading dataset
from google.colab import files

# To suppress scientific notations for a dataframe
pd.set_option("display.float_format", lambda x: "%.4f" % x)

# To suppress warnings
import warnings

warnings.filterwarnings("ignore")
```

1.1.7 Let us load the data

[2]:
```
# let colab access my google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
data = pd.read_csv('/content/drive/MyDrive/used_cars.csv')
```

1.2 Data Overview

Lets do some data sanity checks from uploaded file, such as:

- Data structure and data types
- Number of missing values and duplicated ones

[4]:
```
df = data.copy()
```

[5]:
```
# First find out the number of rows and columns in the data
df.shape
```

[5]: (7253, 14)

[6]:
```python
#Lets get some insights about the data types.
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   S.No.              7253 non-null   int64
 1   Name               7253 non-null   object
 2   Location           7253 non-null   object
 3   Year               7253 non-null   int64
 4   Kilometers_Driven  7253 non-null   int64
 5   Fuel_Type          7253 non-null   object
 6   Transmission       7253 non-null   object
 7   Owner_Type         7253 non-null   object
 8   Mileage            7251 non-null   float64
 9   Engine             7207 non-null   float64
 10  Power              7078 non-null   float64
 11  Seats              7200 non-null   float64
 12  New_price          1006 non-null   float64
 13  Price              6019 non-null   float64
dtypes: float64(6), int64(3), object(5)
memory usage: 793.4+ KB
None
```

Observation

- There are some categorical features
- There are some missing values we need to find out.

[7]:
```python
print(df.isnull().sum())
```

```
S.No.                   0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 46
Power                 175
Seats                  53
New_price            6247
Price                1234
```

```
dtype: int64
```

[8]:
```
pd.DataFrame({'Count':df.isnull().sum()[df.isnull().sum()>0],'Percentage':(df.
↪isnull().sum()[df.isnull().sum()>0]/df.shape[0])*100})
```

[8]:

	Count	Percentage
Mileage	2	0.0276
Engine	46	0.6342
Power	175	2.4128
Seats	53	0.7307
New_price	6247	86.1299
Price	1234	17.0136

Observation:

- **Mileage**, **Engine**, and **Seats** have less than 1% missing data and **Power** slightly has more than 2% missing data. Overall these features have moderate missing data, we can handle them later.
- **New__price** has 6247 missing rows which is more than 86% of its data, and it is very large number.
- **Price** has 1234 missing data which is about 17% of its rows.

[9]:
```
#Check duplicate rows
print(df.duplicated().sum())
```

```
0
```

[10]:
```
# First 10 rows
df.head(20)
```

[10]:

	S.No.	Name	Location	Year \
0	0	Maruti Wagon R LXI CNG	Mumbai	2010
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015
2	2	Honda Jazz V	Chennai	2011
3	3	Maruti Ertiga VDI	Chennai	2012
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013
5	5	Hyundai EON LPG Era Plus Option	Hyderabad	2012
6	6	Nissan Micra Diesel XV	Jaipur	2013
7	7	Toyota Innova Crysta 2.8 GX AT 8S	Mumbai	2016
8	8	Volkswagen Vento Diesel Comfortline	Pune	2013
9	9	Tata Indica Vista Quadrajet LS	Chennai	2012
10	10	Maruti Ciaz Zeta	Kochi	2018
11	11	Honda City 1.5 V AT Sunroof	Kolkata	2012
12	12	Maruti Swift VDI BSIV	Jaipur	2015
13	13	Land Rover Range Rover 2.2L Pure	Delhi	2014
14	14	Land Rover Freelander 2 TD4 SE	Pune	2012
15	15	Mitsubishi Pajero Sport 4X4	Delhi	2014
16	16	Honda Amaze S i-Dtech	Kochi	2016
17	17	Maruti Swift DDiS VDI	Jaipur	2017

```
18     18              Renault Duster 85PS Diesel RxL Plus       Kochi  2014
19     19  Mercedes-Benz New C-Class C 220 CDI BE Avantgare   Bangalore  2014

    Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage    Engine  \
0               72000       CNG       Manual      First  26.6000  998.0000
1               41000    Diesel       Manual      First  19.6700 1582.0000
2               46000    Petrol       Manual      First  18.2000 1199.0000
3               87000    Diesel       Manual      First  20.7700 1248.0000
4               40670    Diesel    Automatic     Second  15.2000 1968.0000
5               75000       LPG       Manual      First  21.1000  814.0000
6               86999    Diesel       Manual      First  23.0800 1461.0000
7               36000    Diesel    Automatic      First  11.3600 2755.0000
8               64430    Diesel       Manual      First  20.5400 1598.0000
9               65932    Diesel       Manual     Second  22.3000 1248.0000
10              25692    Petrol       Manual      First  21.5600 1462.0000
11              60000    Petrol    Automatic      First  16.8000 1497.0000
12              64424    Diesel       Manual      First  25.2000 1248.0000
13              72000    Diesel    Automatic      First  12.7000 2179.0000
14              85000    Diesel    Automatic     Second   0.0000 2179.0000
15             110000    Diesel       Manual      First  13.5000 2477.0000
16              58950    Diesel       Manual      First  25.8000 1498.0000
17              25000    Diesel       Manual      First  28.4000 1248.0000
18              77469    Diesel       Manual      First  20.4500 1461.0000
19              78500    Diesel    Automatic      First  14.8400 2143.0000

      Power  Seats  New_price   Price
0   58.1600 5.0000        NaN  1.7500
1  126.2000 5.0000        NaN 12.5000
2   88.7000 5.0000     8.6100  4.5000
3   88.7600 7.0000        NaN  6.0000
4  140.8000 5.0000        NaN 17.7400
5   55.2000 5.0000        NaN  2.3500
6   63.1000 5.0000        NaN  3.5000
7  171.5000 8.0000    21.0000 17.5000
8  103.6000 5.0000        NaN  5.2000
9   74.0000 5.0000        NaN  1.9500
10 103.2500 5.0000    10.6500  9.9500
11 116.3000 5.0000        NaN  4.4900
12  74.0000 5.0000        NaN  5.6000
13 187.7000 5.0000        NaN 27.0000
14 115.0000 5.0000        NaN 17.5000
15 175.5600 7.0000    32.0100 15.0000
16  98.6000 5.0000        NaN  5.4000
17  74.0000 5.0000        NaN  5.9900
18  83.8000 5.0000        NaN  6.3400
19 167.6200 5.0000        NaN 28.0000
```

Observations:

- We can see the **S.No.** column is like index column which can be dropped from data set.
- We can see there is a row 14 there is a car with year 2012 and with kilometers_driven = 85000 which has **0.0 Mileage** which can be a data entry issue. We need to find out if there can be more like this in Mileage.
- **Name** column has combination of the *Brand* and *Models* of the cars which can be separated into different sets.

1.3 Exploratory Data Analysis

First we check statistical information of all features

[11]:
```
df.describe(include='all')
```

[11]:
```
             S.No.                    Name Location      Year  \
count   7253.0000                    7253     7253 7253.0000
unique        NaN                    2041       11       NaN
top           NaN  Mahindra XUV500 W8 2WD   Mumbai       NaN
freq          NaN                      55      949       NaN
mean    3626.0000                     NaN      NaN 2013.3654
std     2093.9051                     NaN      NaN    3.2544
min        0.0000                     NaN      NaN 1996.0000
25%     1813.0000                     NaN      NaN 2011.0000
50%     3626.0000                     NaN      NaN 2014.0000
75%     5439.0000                     NaN      NaN 2016.0000
max     7252.0000                     NaN      NaN 2019.0000

        Kilometers_Driven Fuel_Type Transmission Owner_Type   Mileage  \
count           7253.0000      7253         7253       7253 7251.0000
unique                NaN         5            2          4       NaN
top                   NaN    Diesel       Manual      First       NaN
freq                  NaN      3852         5204       5952       NaN
mean           58699.0631       NaN          NaN        NaN   18.1416
std            84427.7206       NaN          NaN        NaN    4.5622
min              171.0000       NaN          NaN        NaN    0.0000
25%            34000.0000       NaN          NaN        NaN   15.1700
50%            53416.0000       NaN          NaN        NaN   18.1600
75%            73000.0000       NaN          NaN        NaN   21.1000
max          6500000.0000       NaN          NaN        NaN   33.5400

           Engine     Power     Seats  New_price     Price
count   7207.0000 7078.0000 7200.0000  1006.0000 6019.0000
unique        NaN       NaN       NaN        NaN       NaN
top           NaN       NaN       NaN        NaN       NaN
freq          NaN       NaN       NaN        NaN       NaN
mean    1616.5735  112.7652    5.2804    22.7797    9.4795
std      595.2851   53.4936    0.8093    27.7593   11.1879
min       72.0000   34.2000    2.0000     3.9100    0.4400
25%     1198.0000   75.0000    5.0000     7.8850    3.5000
```

```
50%     1493.0000   94.0000    5.0000    11.5700    5.6400
75%     1968.0000  138.1000    5.0000    26.0425    9.9500
max     5998.0000  616.0000   10.0000   375.0000  160.0000
```

Statistical Analysis

- We can observe that there are 5 categorical features.
- **Name** has 2041 unique values with "Mahindra XUV500 W8 2WD" as most frequent (or popular) car in this category.
- **Location** with 11 unique value where *Mumbai* has the most used cars.
- **Kilometers_Driven** has 75% of the used cars have less than *73000km* but there is a max value of *6500000km*. We need to find out why.
- **Power** has 75% of car 138 or less with max value about 616.
- **Price** same as *Power* has 75% of its value below 9.95 and a max jump to 160.
- **Engine** follows the same pattern than *Price* & *Power*. So their might be a correlation there.
- Other numerical fatures are relatively in correct data range.

Lets find out about the Kilometers_Driven max value

[12]:
```python
df.sort_values(by='Kilometers_Driven', ascending=False, axis=0,).head()
```

[12]:
```
      S.No.                                    Name Location  Year  \
2328   2328                BMW X5 xDrive 30d M Sport  Chennai  2017
340     340         Skoda Octavia Ambition Plus 2.0 TDI AT  Kolkata  2013
1860   1860             Volkswagen Vento Diesel Highline  Chennai  2013
358     358                      Hyundai i10 Magna 1.2  Chennai  2009
2823   2823  Volkswagen Jetta 2013-2015 2.0L TDI Highline AT  Chennai  2015

      Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage    Engine  \
2328            6500000    Diesel    Automatic      First  15.9700 2993.0000
340              775000    Diesel    Automatic      First  19.3000 1968.0000
1860             720000    Diesel       Manual      First  20.5400 1598.0000
358              620000    Petrol       Manual      First  20.3600 1197.0000
2823             480000    Diesel    Automatic      First  16.9600 1968.0000

        Power  Seats  New_price    Price
2328 258.0000 5.0000        NaN  65.0000
340  141.0000 5.0000        NaN   7.5000
1860 103.6000 5.0000        NaN   5.9000
358   78.9000 5.0000        NaN   2.7000
2823 138.0300 5.0000        NaN  13.0000
```

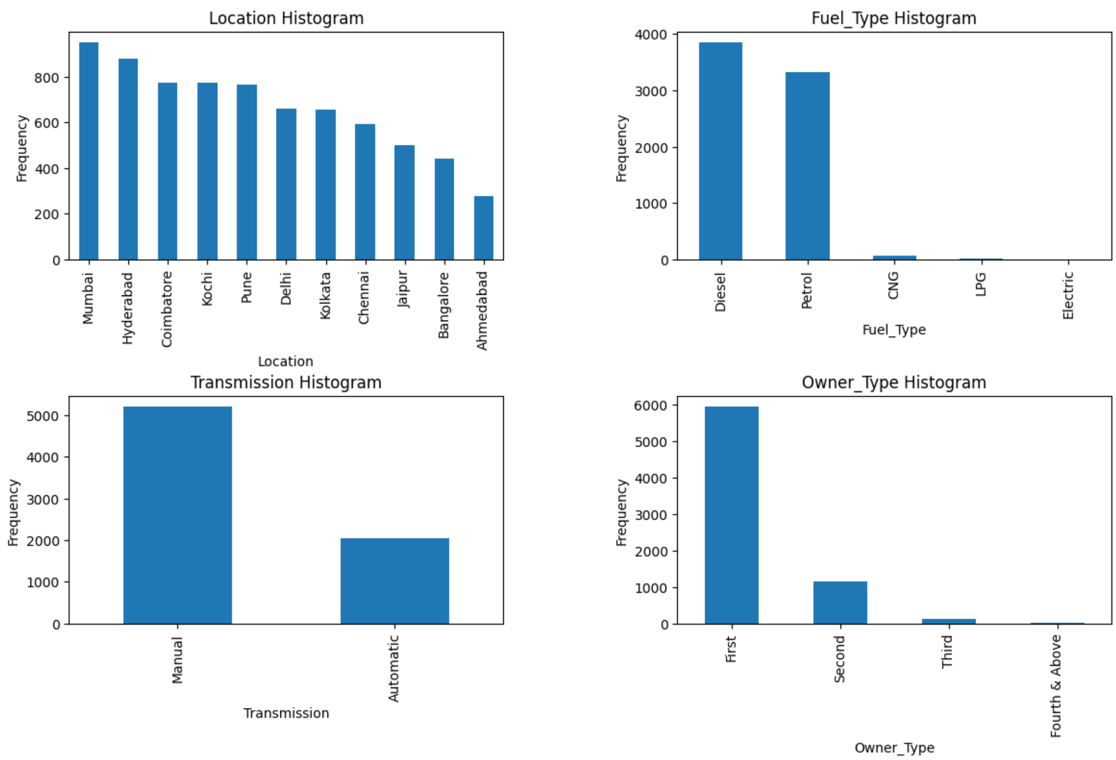
We can drop the row 2328 since it is not fitting the correct data

[13]:
```python
#Drop the row of Kilometer_Driven with wrong value
df.drop(axis=0, index=2328, inplace=True)
```

1.4 Univariate Analysis

Starting by categorical features

```
[14]: cat_cols = df.drop('Name', axis=1).select_dtypes(include = ['object']).columns

fig = plt.figure(figsize=(14, 8))
fig.subplots_adjust(hspace=0.6, wspace=0.4)

for name in cat_cols:
  plt.subplot(2, 2, cat_cols.get_loc(name)+1)
  plt1 = df[name].value_counts().plot(kind='bar')
  plt.title(name+' Histogram')
  plt1.set(xlabel = name, ylabel='Frequency')

plt.show()
```

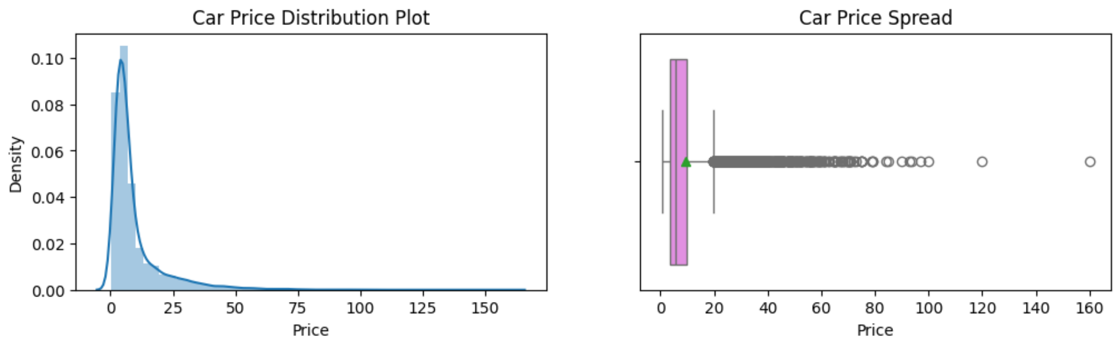


Observation:

- Locations Mumbai and Hydarabad have the highest car sales following by other locations.
- Fule_type Diesel and Petrol have the significantly highest sales than the other fule types models.
- Manual transmission has about 75% of the care sales.
- Owner_type by first owner has almost 90% of the car sales.

Continue now with numerical features

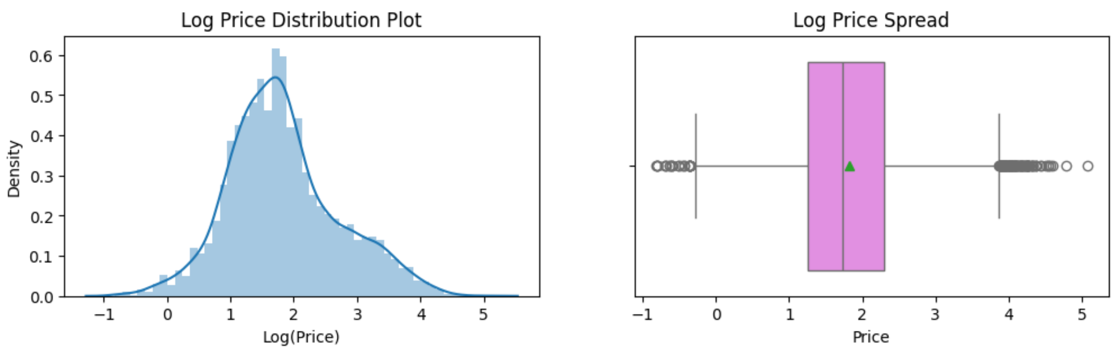
[15]:
```
# Start with Price feature
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.distplot(df['Price'])
plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(x=df['Price'], showmeans = True, color = "violet")
plt.show()
```



Observations:

- We can see most used care price are less than 25
- We see its density plot is right skewed which can be due to some outlier that we can see in the boxplot as well. -Outliers can be the high end cars like Bently and Lamburghinie.

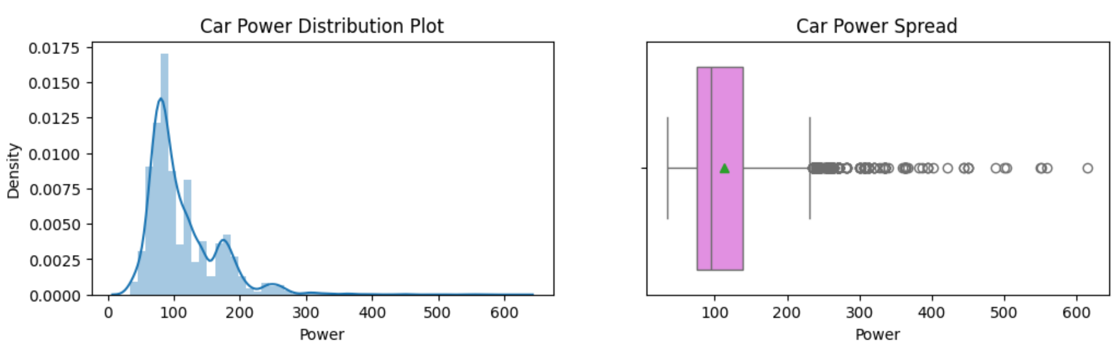
[16]:
```
# Log transformation of the feature 'Price'
# Start with Price feature
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Log Price Distribution Plot')
sns.distplot(np.log(df["Price"]), axlabel = "Log(Price)");
plt.subplot(1,2,2)
plt.title('Log Price Spread')
sns.boxplot(x=np.log(df["Price"]), showmeans = True, color = "violet")
plt.show()
```



Since feature ‘Price’ is highly right skewed, therefor we can use its log transformation for our data analysis by adding a new column as ‘price_log’.

[17]:
```
df['price_log'] = np.log(df['Price'])
```

[18]:
```
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Car Power Distribution Plot')
sns.distplot(df['Power'])
plt.subplot(1,2,2)
plt.title('Car Power Spread')
sns.boxplot(x=df['Power'], showmeans = True, color = "violet")
plt.show()
```

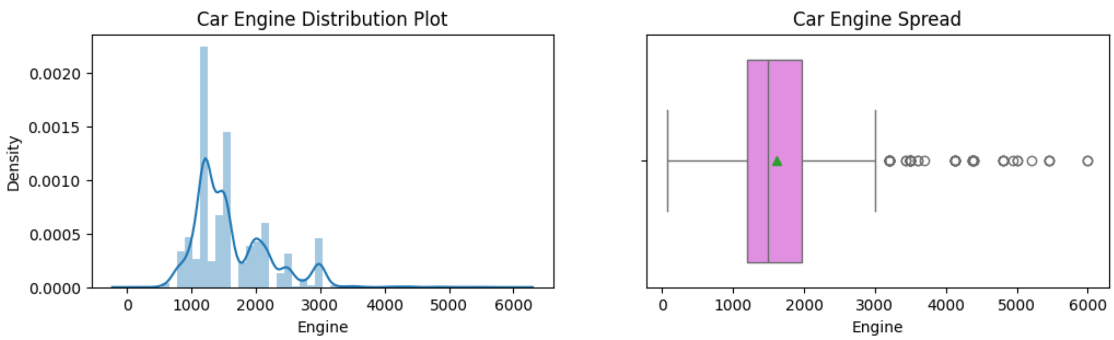


Observations:

- We can see most used care Power are less than 200
- We see its density plot is right skewed which can be due to some outlier that we can see in the boxplot as well.

[19]:
```
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Car Engine Distribution Plot')
```

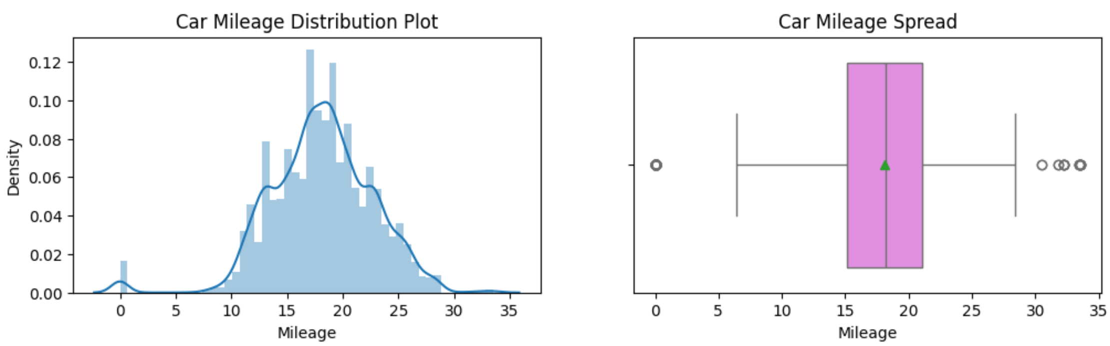
```
sns.distplot(df['Engine'])
plt.subplot(1,2,2)
plt.title('Car Engine Spread')
sns.boxplot(x=df['Engine'], showmeans = True, color = "violet")
plt.show()
```



Observations:

- We can see most used care Engine size are less than 3000
- We see also its density plot is right skewed which can be due to some outlier that we can see in the boxplot as well.

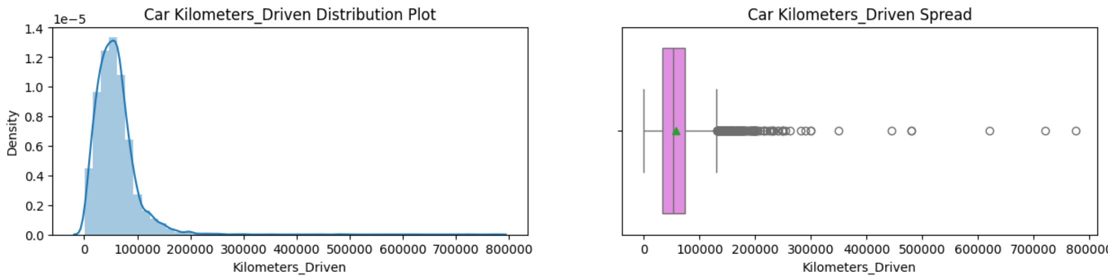
[20]:
```
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Car Mileage Distribution Plot')
sns.distplot(df['Mileage'])
plt.subplot(1,2,2)
plt.title('Car Mileage Spread')
sns.boxplot(x=df['Mileage'], showmeans = True, color = "violet")
plt.show()
```



Observation:

Mileage looks like more a normal distribution with slightly left skewed and some outlier on the right side which can be due to sports care engines.

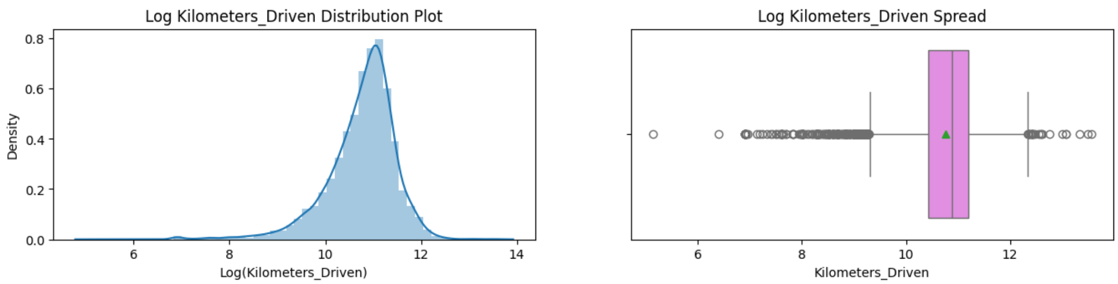
[21]:
```
plt.figure(figsize=(15,3))
plt.subplot(1,2,1)
plt.title('Car Kilometers_Driven Distribution Plot')
sns.distplot(df['Kilometers_Driven'])
plt.subplot(1,2,2)
plt.title('Car Kilometers_Driven Spread')
sns.boxplot(x=df['Kilometers_Driven'], showmeans = True, color = "violet")
plt.show()
```



Observation

- Kilometers_Driven is highly right-skewed.
- And there are still outliers.
- We can use log transformation.

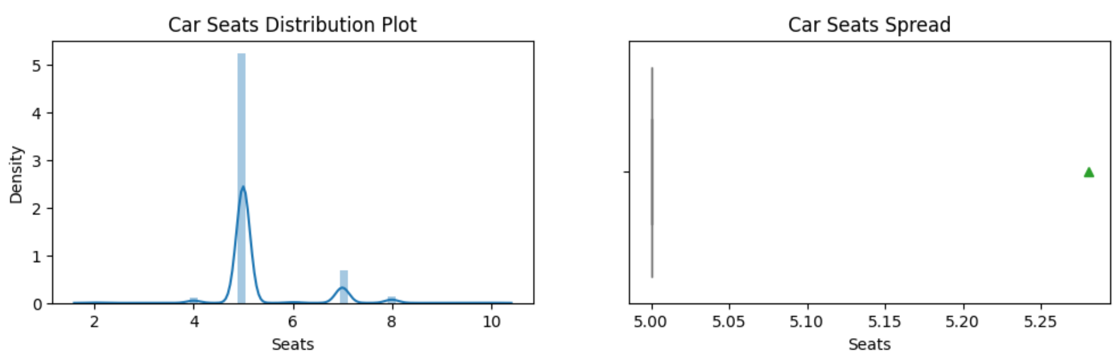
[22]:
```
# Log transformation of the feature 'Kilometers_Driven'
plt.figure(figsize=(15,3))
plt.subplot(1,2,1)
plt.title('Log Kilometers_Driven Distribution Plot')
sns.distplot(np.log(df["Kilometers_Driven"]), axlabel =␣
 ↪"Log(Kilometers_Driven)");
plt.subplot(1,2,2)
plt.title('Log Kilometers_Driven Spread')
sns.boxplot(x=np.log(df["Kilometers_Driven"]), showmeans = True, color =␣
 ↪"violet")
plt.show()
```



Since feature 'Kilometers_driven' is highly right skewed, therefor we can use its log transformation for our data analysis. Add a new column for 'Kilometers_driven_log'

[23]:
```
df['Kilometers_driven_log'] = np.log(df['Kilometers_Driven'])
```

[24]:
```
plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.title('Car Seats Distribution Plot')
sns.distplot(df['Seats'])
plt.subplot(1,2,2)
plt.title('Car Seats Spread')
sns.boxplot(x=df['Seats'], showfliers=False, showmeans = True, color = "violet")
plt.show()
```

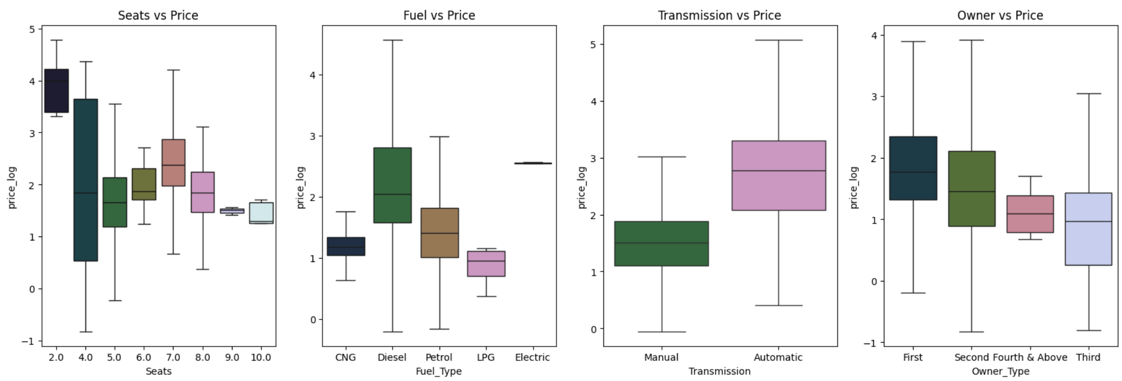


1.5 Bivariate Analysis

First lets check categorical features vs. Price feature

[25]:
```
plt.figure(figsize=(20,6))

plt.subplot(1,4,1)
plt.title('Seats vs Price')
```

```
sns.boxplot(x=df['Seats'], y=df['price_log'],␣
↪palette=("cubehelix"),showfliers=False)

plt.subplot(1,4,2)
plt.title('Fuel vs Price')
sns.boxplot(x=df['Fuel_Type'], y=df['price_log'],␣
↪palette=("cubehelix"),showfliers=False)

plt.subplot(1,4,3)
plt.title('Transmission vs Price')
sns.boxplot(x=df['Transmission'], y=df['price_log'],␣
↪palette=("cubehelix"),showfliers=False)

plt.subplot(1,4,4)
plt.title('Owner vs Price')
sns.boxplot(x=df['Owner_Type'], y=df['price_log'],␣
↪palette=("cubehelix"),showfliers=False)

plt.show()
```

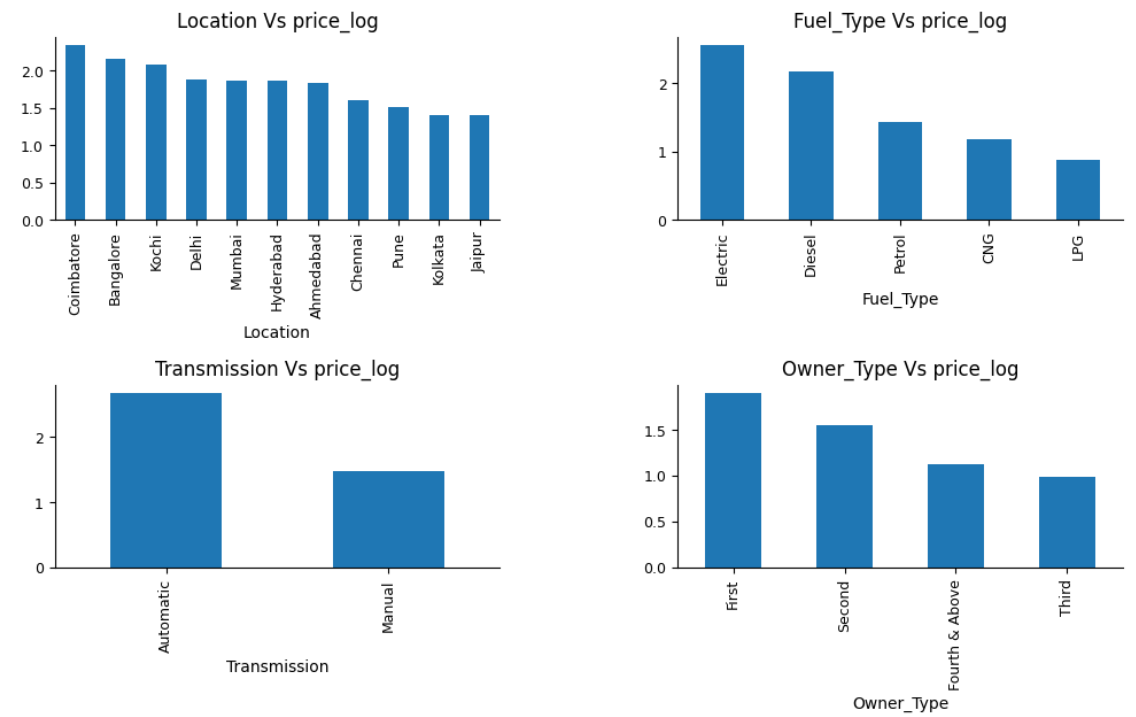


Observation:

- We can see 2 seated cars are most expensive follow by 7.
- 4 seated cars have higher sold, followed by 5, 7 and 8.
- Two seated care are most expensive maybe because thay are sportive models.
- Full_type Diesel cars are most popular more expensive followed by Petrol cars.
- Transmission automatics have higher price.
- First owner cars are more expensive compare to others.

[26]:
```
target = 'price_log'
grid_x = 2
grid_y = 2
name = cat_cols[0]
fig, axarr = plt.subplots(grid_y, grid_x, figsize=(12, 6))
```

```
for i in range(grid_y):
  name = cat_cols[cat_cols.get_loc(name)+i]
  for j in range(grid_x):
    df.groupby(name)[target].mean().sort_values(ascending=False).plot.
 ↪bar(ax=axarr[i][j], fontsize=9)
    axarr[i][j].set_title(name+" Vs "+target, fontsize=12)
    name = cat_cols[cat_cols.get_loc(name)+1-j]
fig.subplots_adjust(hspace=0.9, wspace=0.4)
sns.despine()
plt.show()
```



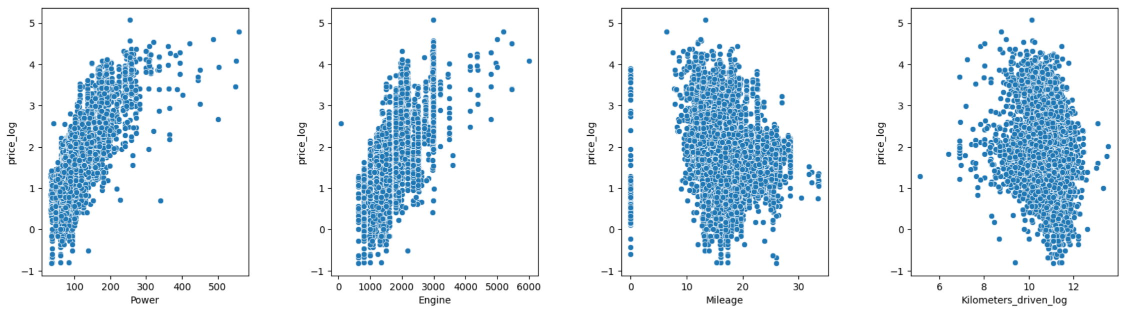
Observation:

- By location we can see most expensive cars are sold in Coimbatore follow by Bangalor and Kochi.
- We can see the electric cars are the most expensive ones and LPG full_type or the least expensive cars.
- Transmission type automatic is more expensive.
- Owner_type first are more expensive cars as well.

Lets compare numeric features using scatter plots

[27]:
```
fig = plt.figure(figsize = (20,5))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
```

```python
ax = fig.add_subplot(1, 4, 1)
sns.scatterplot(data=df, x="Power", y="price_log")

ax = fig.add_subplot(1, 4, 2)
sns.scatterplot(data=df, x="Engine", y="price_log");

ax = fig.add_subplot(1, 4, 3)
sns.scatterplot(data=df, x="Mileage", y="price_log");

ax = fig.add_subplot(1, 4, 4)
sns.scatterplot(data=df, x="Kilometers_driven_log", y="price_log");

plt.show()
```

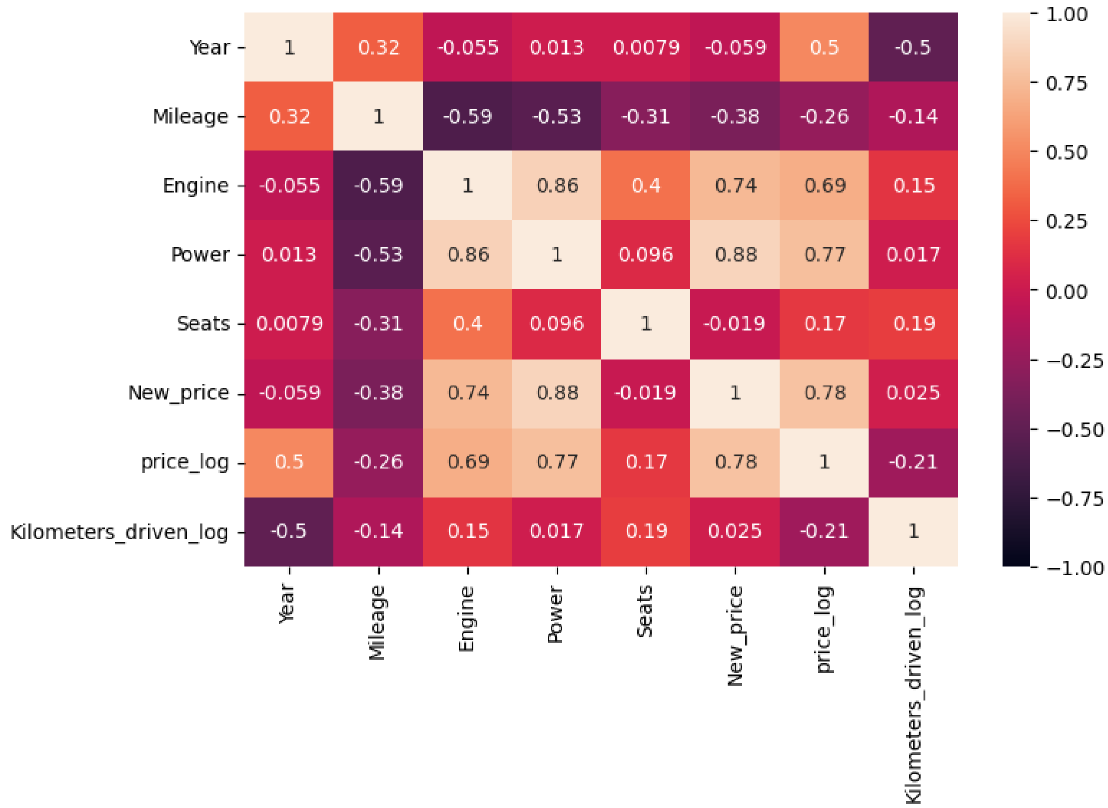


Observation:

- **Power** & **Price** follow some patterns when Power increases Price increases as well.
- **Engine** & **Price** follow the same pattern than Power and Price.
- **Mileage** & **Price** are not following a pattern.
- **Kilometers_driven** & **Price** are not following a pattern.

1.5.1 HeatMap Analysis

[28]:
```python
plt.figure(figsize = (8, 5))
sns.heatmap(df.drop(['Kilometers_Driven', 'Price', 'S.No.'], axis=1).
 ↪corr(numeric_only = True), annot = True, vmin = -1, vmax = 1)
plt.show()
```



Observation:

- **Kilometer__driven__log** has a negative correlation with year.
- **Price** & **Kilometer__driven__log** have negative correlation.
- **Engine** has a strong positive correlation with Power 0.86
- **Price** has a positive correlation with Engine 0.69 as well with Power 0.77
- **Mileage** has negative correlation with Engine, Power, and Price.
- **Price__log** has moderate positive correlation with year.

1.5.2 Feature Engineering

The `Name` column in the current format might not be very useful in our analysis. Since the name contains both the brand name and the model name of the vehicle, the column would have too many unique values to be useful in prediction. We can extract that information from that column.

[29]:
```
new_df = df.copy()
```

[30]:
```
#Splitting company name from CarName column
brands = df['Name'].apply(lambda x : x.split(' ')[0])
models = df['Name'].apply(lambda x : x.split(' ')[1:]).str.join(' ')
brands = brands.str.lower()
brands.replace('land','land-rover',inplace=True)
```

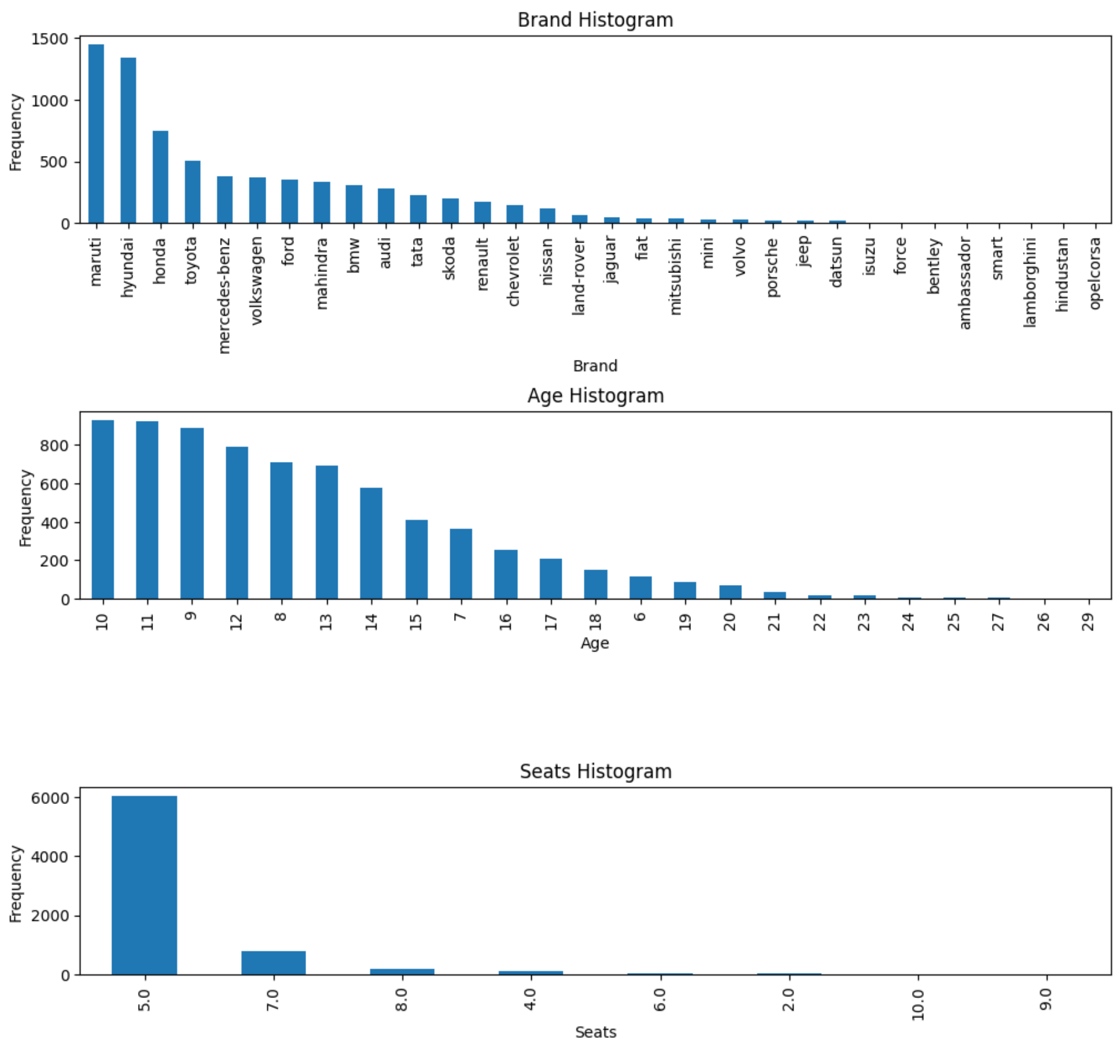
```
print("Number of brands", brands.unique().shape[0])
print("Number of models", models.unique().shape[0])
print("Unique brands:", brands.unique())
```

```
Number of brands 32
Number of models 2041
Unique brands: ['maruti' 'hyundai' 'honda' 'audi' 'nissan' 'toyota' 'volkswagen'
'tata'
 'land-rover' 'mitsubishi' 'renault' 'mercedes-benz' 'bmw' 'mahindra'
 'ford' 'porsche' 'datsun' 'jaguar' 'volvo' 'chevrolet' 'skoda' 'mini'
 'fiat' 'jeep' 'smart' 'ambassador' 'isuzu' 'force' 'bentley'
 'lamborghini' 'hindustan' 'opelcorsa']
```

[31]:
```
new_df['Brand'] = brands
```

[32]:
```
from datetime import date
date.today().year
new_df['Age']=date.today().year-df['Year']
```

[33]:
```
cat_cols = ['Brand', 'Age', 'Seats']
fig = plt.figure(figsize=(12, 11))
fig.subplots_adjust(hspace=1.0, wspace=0.4)

i = 1
for name in cat_cols:
  plt.subplot(3, 1, i)
  plt1 = new_df[name].value_counts().plot(kind='bar')
  plt.title(name+' Histogram')
  plt1.set(xlabel = name, ylabel='Frequency')
  i+=1
plt.show()
```

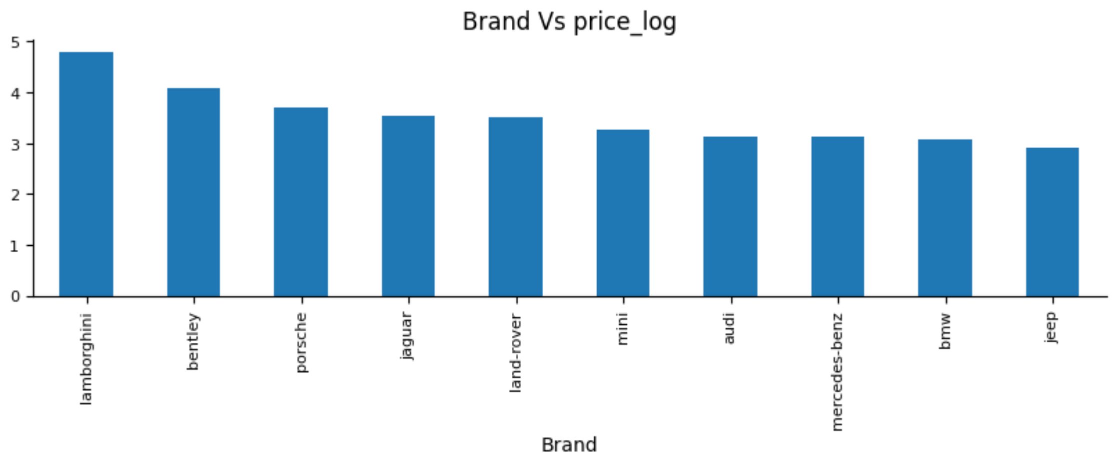


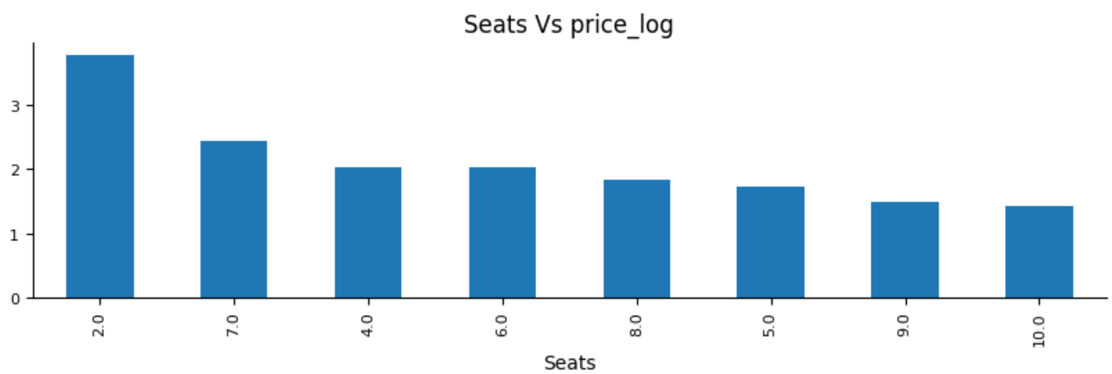
Observation:

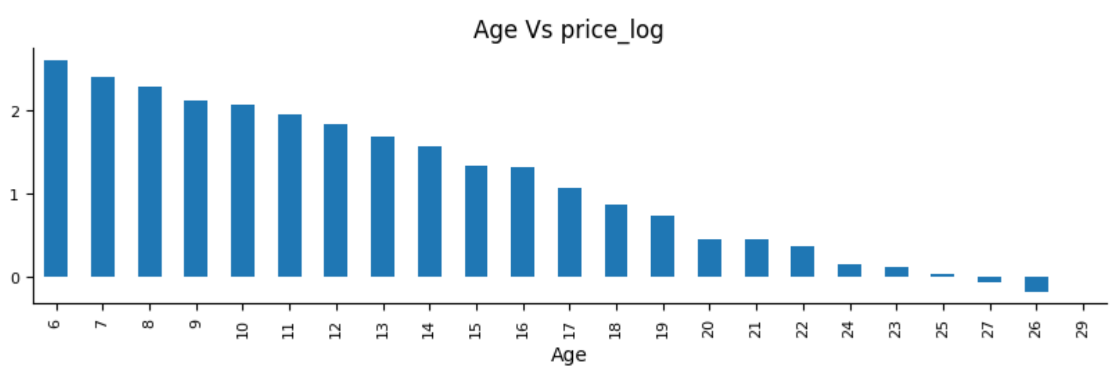
- **Brands** '*maruti*' and '*hyundai*' are the most sold cars combine almost 50% of the cars sold, followed by 'honda' around 10% of cars sold.
- **Age** of the most sold cars are between 8 to 14 years.

[34]:
```
fig, axarr = plt.subplots( 3, figsize=(10, 11))
new_df.groupby('Brand')['price_log'].mean().sort_values(ascending=False).
 ↪head(10).plot.bar(ax=axarr[0], fontsize=8)
axarr[0].set_title("Brand Vs price_log", fontsize=12)
new_df.groupby('Seats')['price_log'].mean().sort_values(ascending=False).plot.
 ↪bar(ax=axarr[1], fontsize=8)
axarr[1].set_title("Seats Vs price_log", fontsize=12)
new_df.groupby('Age')['price_log'].mean().sort_values(ascending=False).plot.
 ↪bar(ax=axarr[2], fontsize=8)
```

```
axarr[2].set_title("Age Vs price_log", fontsize=12)
plt.subplots_adjust(hspace=0.8)
plt.subplots_adjust(wspace=.5)
sns.despine()
```







Observation:

- Lamborghini brand is the most expensive care following by Bently and Porsche.
- Two seated are are the most expensive. It can be due to their expensive brand.

- New cars are more expensive and the price decrease by Age increases.

1.5.3 Missing value treatment

Handling the missing values

Price missing Values

[35]:
```
new_df.dropna(subset=['price_log'], inplace=True)
new_df.shape
```

[35]:
```
(6018, 18)
```

Mileage zero values

We saw there might be zero values in mileage feature which can be data error when they are added to data set we need to handle them as **Nan** values,

[36]:
```
new_df.loc[new_df["Mileage"]==0.0,'Mileage']=np.nan
print(new_df.Mileage.isnull().sum())
```

```
70
```

[37]:
```
new_df['Mileage'].fillna(value=np.mean(new_df['Mileage']),inplace=True)
print(new_df.Mileage.isnull().sum())
```

```
0
```

We chose to impute **Mileage** missing values by the mean value because the mean and median values are almost the same for this feature.

Power missing values

[38]:
```
print(new_df.Power.isnull().sum())
new_df['Power'] = new_df.groupby(['Brand'])['Power'].transform(lambda x:x.
 ↪fillna(x.median()))
```

```
143
```

[39]:
```
print(new_df.Power.isnull().sum())
```

```
1
```

[40]:
```
new_df.dropna(subset=['Power'], inplace=True)
print(new_df.Power.isnull().sum())
```

```
0
```

Power is right skewed and we can median for imputing the missing values

Seats missing values

[41]:
```
print(new_df.Seats.isnull().sum())
```

```
new_df['Seats'] = new_df.groupby(['Brand'])['Seats'].transform(lambda x:x.
    ↪fillna(x.median()))
```

42

[42]:
```
print(new_df.Seats.isnull().sum())
```

0

Engine missing values

[43]:
```
print(new_df.Engine.isnull().sum())
new_df['Engine'] = new_df.groupby(['Brand'])['Engine'].transform(lambda x:x.
    ↪fillna(x.median()))
```

36

[44]:
```
print(new_df.Engine.isnull().sum())
```

0

Engine are regrouped by Brand and we can use median for imputing the missing values.

[45]:
```
new_df.drop(['New_price'], axis=1, inplace=True)
```

1.6 Important Insights from EDA and Data Preprocessing

- Most of the customers prefer 5 Seats cars and followed by 7 seats.
- The price of the 2-seat cars is higher than other cars because they are sportive cars.
- The price of the car decreases as the Age of the car increases and its Kilometer_diven or Mileage naturally increase as well.
- First owner cars are preferred by customer rather than the Second or Third.
- The customers prefers to purchase an Diesel fule type cars maybe because they are cheaper and Diesel is less expensive in India.
- Manual Transmission cars are cheaper, hence more popular than automatic.

1.7 Building Various Models

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.

1.7.1 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.

[46]:
```
new_df.drop(['Name', 'Year', 'Price', 'Kilometers_Driven', 'S.No.'], axis=1,␣
↪inplace=True)
```

[47]:
```
new_df.shape
```

[47]: (6017, 12)

[48]:
```
Y = new_df['price_log']
X = new_df.drop(['price_log'], axis = 1)
```

[49]:
```
X_dummies = pd.get_dummies(X, columns=['Location', 'Fuel_Type', 'Transmission',␣
↪'Owner_Type', 'Brand'], dtype=int, drop_first=True)
X_dummies.shape
```

[49]: (6017, 52)

[50]:
```
X_train, X_test, y_train, y_test = train_test_split(X_dummies, Y, test_size = 0.
↪3, random_state = 1)
print(X_train.shape, X_test.shape)
```

(4211, 52) (1806, 52)

[51]:
```
# Function to compute adjusted R-squared
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))

# Function to compute different metrics to check performance of a regression␣
↪model
def model_performance_regression(model, predictors, target):
    """
    Function to compute different metrics to check regression model performance

    model: regressor
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    r2 = r2_score(target, pred)                          # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)     # to compute adjusted␣
↪R-squared
```

```
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)           # to compute MAE

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "RMSE": rmse,
            "MAE": mae,
            "R-squared": r2,
            "Adj. R2": adjr2,
        },
        index=[0],
    )

    return df_perf
```

1.8 Regression Models Building

Using following various regression algorithems

1) Linear Regression

2) Ridge / Lasso Regression

3) Decision Trees

4) Random Forest

1.8.1 Linear Regression

[52]:
```
# Normaliser les x_train et x_test
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[53]:
```
# Create a linear regression model
lr = LinearRegression()
# Fit linear regression model
lr.fit(X_train_scaled, y_train)
# Get score of the model
dtree_model_train_perf = model_performance_regression(lr, X_train_scaled,␣
 ↪y_train)
dtree_model_train_perf
```

[53]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2378	0.1777	0.9269	0.9260

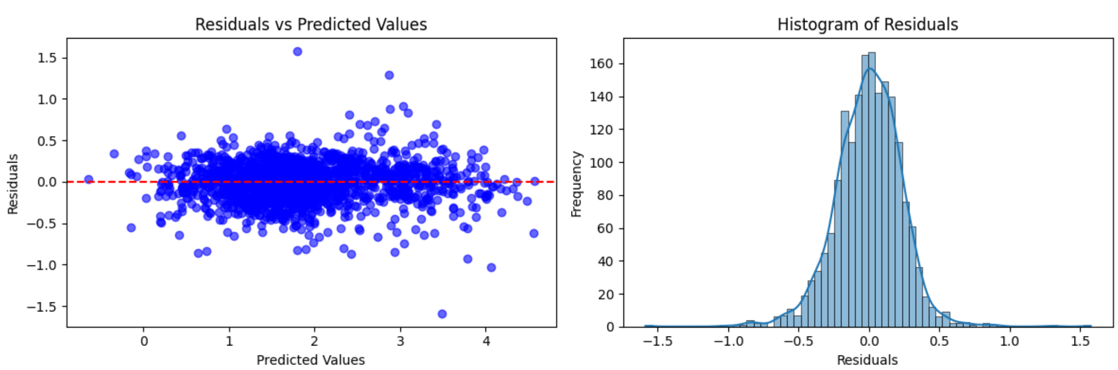
[54]:
```
lreg_model_test_perf = model_performance_regression(lr, X_test_scaled, y_test)
lreg_model_test_perf
```

[54]:
```
     RMSE     MAE  R-squared  Adj. R2
0  0.2354  0.1778     0.9248   0.9226
```

Observation:

We can see that Linear model represent a good scores. Both train set and test set produce almost same scores which is good indication that our model doesn't underfit or overfit.

[55]:
```python
#Getting prediction values for both train and test sets
y_train_pred = lr.predict(X_train_scaled)
y_test_pred = lr.predict(X_test_scaled)

#Calculating residuels
res = y_test - y_test_pred

# Scatter plot of residuals
plt.figure(figsize=(12, 4))

# Scatter plot of residuals vs predicted values
plt.subplot(1, 2, 1)
plt.scatter(y_test_pred, res, color='blue', alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')

# Histogram of residuals
plt.subplot(1, 2, 2)
sns.histplot(res, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')

# Show the plots
plt.tight_layout()
plt.show()
```



Observation

- We can see that there is no pattern in the residuals vs fitted values scatter plot and we have our linearity satisfied.
- We can see that the error terms are normally distributed. The assumption of normality is satisfied.

Linear Regression using StatModel library

[56]:
```
X1 = sm.add_constant(X)
X1_dummies = pd.get_dummies(X1, columns=['Location', 'Fuel_Type',␣
↪'Transmission', 'Owner_Type', 'Brand'], dtype=int, drop_first=True)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1_dummies, Y,␣
↪test_size = 0.3, random_state = 1)
print(X1_train.shape, X1_test.shape)
```

```
(4211, 53) (1806, 53)
```

[57]:
```
# create OLS model
ols_model = sm.OLS(y1_train, X1_train).fit()

# get the model summary
print(ols_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              price_log   R-squared:                       0.927
Model:                            OLS   Adj. R-squared:                  0.926
Method:                 Least Squares   F-statistic:                     1015.
Date:                Tue, 15 Apr 2025   Prob (F-statistic):               0.00
Time:                        14:10:07   Log-Likelihood:                 73.939
No. Observations:                4211   AIC:                            -41.88
Df Residuals:                    4158   BIC:                             294.4
Df Model:                          52
Covariance Type:            nonrobust
================================================================================
=============
                                coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-------------
const                         3.2130      0.260     12.335      0.000
2.702       3.724
Mileage                      -0.0156      0.002     -9.034      0.000
-0.019      -0.012
Engine                        0.0002   2.01e-05      8.081      0.000
0.000       0.000
Power                         0.0051      0.000     23.817      0.000
```

```
0.005       0.005
Seats                          0.0461      0.007      6.337      0.000
0.032       0.060
Kilometers_driven_log         -0.0668      0.007     -9.700      0.000
-0.080      -0.053
Age                           -0.1196      0.002    -72.696      0.000
-0.123      -0.116
Location_Bangalore             0.1664      0.024      6.813      0.000
0.119       0.214
Location_Chennai               0.0342      0.023      1.478      0.139
-0.011       0.080
Location_Coimbatore            0.0978      0.022      4.381      0.000
0.054       0.142
Location_Delhi                -0.0584      0.023     -2.576      0.010
-0.103      -0.014
Location_Hyderabad             0.1224      0.022      5.631      0.000
0.080       0.165
Location_Jaipur               -0.0698      0.024     -2.947      0.003
-0.116      -0.023
Location_Kochi                -0.0284      0.022     -1.270      0.204
-0.072       0.015
Location_Kolkata              -0.2340      0.023    -10.292      0.000
-0.279      -0.189
Location_Mumbai               -0.0466      0.022     -2.144      0.032
-0.089      -0.004
Location_Pune                 -0.0369      0.022     -1.654      0.098
-0.081       0.007
Fuel_Type_Diesel               0.2170      0.040      5.431      0.000
0.139       0.295
Fuel_Type_Electric             1.1024      0.244      4.526      0.000
0.625       1.580
Fuel_Type_LPG                 -0.1359      0.099     -1.367      0.172
-0.331       0.059
Fuel_Type_Petrol              -0.0705      0.041     -1.720      0.085
-0.151       0.010
Transmission_Manual           -0.1103      0.012     -9.006      0.000
-0.134      -0.086
Owner_Type_Fourth & Above      0.0726      0.086      0.849      0.396
-0.095       0.240
Owner_Type_Second             -0.0701      0.011     -6.364      0.000
-0.092      -0.049
Owner_Type_Third              -0.1193      0.028     -4.289      0.000
-0.174      -0.065
Brand_audi                     0.4507      0.243      1.856      0.064
-0.025       0.927
Brand_bentley                 -0.0061      0.347     -0.018      0.986
-0.687       0.674
Brand_bmw                      0.4320      0.243      1.778      0.075
```

```
-0.044       0.908
Brand_chevrolet                 -0.4401      0.243     -1.813      0.070
-0.916       0.036
Brand_datsun                    -0.6129      0.254     -2.416      0.016
-1.110      -0.116
Brand_fiat                      -0.4684      0.248     -1.886      0.059
-0.955       0.018
Brand_force                     -0.0175      0.295     -0.059      0.953
-0.596       0.561
Brand_ford                      -0.2179      0.242     -0.901      0.368
-0.692       0.256
Brand_honda                     -0.0758      0.242     -0.313      0.754
-0.550       0.399
Brand_hyundai                   -0.1315      0.242     -0.544      0.587
-0.606       0.343
Brand_isuzu                     -0.3661      0.279     -1.314      0.189
-0.912       0.180
Brand_jaguar                     0.4347      0.246      1.765      0.078
-0.048       0.918
Brand_jeep                       0.0633      0.254      0.250      0.803
-0.434       0.561
Brand_lamborghini                0.4185      0.346      1.208      0.227
-0.261       1.098
Brand_land-rover                 0.7524      0.245      3.072      0.002
0.272       1.233
Brand_mahindra                  -0.3007      0.242     -1.240      0.215
-0.776       0.175
Brand_maruti                    -0.1383      0.242     -0.572      0.568
-0.612       0.336
Brand_mercedes-benz              0.4729      0.243      1.949      0.051
-0.003       0.949
Brand_mini                       0.8962      0.248      3.619      0.000
0.411       1.382
Brand_mitsubishi                 0.0785      0.246      0.318      0.750
-0.405       0.562
Brand_nissan                    -0.1613      0.243     -0.663      0.508
-0.638       0.316
Brand_porsche                    0.0369      0.254      0.145      0.884
-0.461       0.535
Brand_renault                   -0.1910      0.243     -0.787      0.431
-0.667       0.285
Brand_skoda                     -0.0503      0.243     -0.208      0.836
-0.526       0.425
Brand_tata                      -0.6380      0.242     -2.632      0.009
-1.113      -0.163
Brand_toyota                     0.0616      0.242      0.254      0.799
-0.413       0.536
Brand_volkswagen                -0.1526      0.242     -0.631      0.528
```

```
-0.627      0.322
Brand_volvo                   0.2403      0.252      0.954      0.340
-0.253      0.734
==============================================================================
Omnibus:                     1170.054   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            20417.548
Skew:                          -0.865   Prob(JB):                         0.00
Kurtosis:                      13.648   Cond. No.                     6.12e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.12e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Observations:

We can see that all the numerical independent variables are statistically significant with very low p-values.

[58]:
```
vif_series = pd.Series(
    [variance_inflation_factor(X1_train.values, i) for i in range(X1_train.
 ↪shape[1])],
    index = X1_train.columns,
    dtype = float)

print("VIF Scores: \n\n{}\n".format(vif_series))
```

```
VIF Scores:

const                    4990.8455
Mileage                     3.7413
Engine                     10.7140
Power                       9.7158
Seats                       2.5351
Kilometers_driven_log       1.7892
Age                         2.2084
Location_Bangalore          2.4412
Location_Chennai            2.9403
Location_Coimbatore         3.4656
Location_Delhi              3.0720
Location_Hyderabad          3.7814
Location_Jaipur             2.6888
Location_Kochi              3.4591
Location_Kolkata            3.1573
Location_Mumbai             3.9993
Location_Pune               3.3816
Fuel_Type_Diesel           29.2414
```

```
Fuel_Type_Electric           1.0361
Fuel_Type_LPG                1.2060
Fuel_Type_Petrol            30.6685
Transmission_Manual          2.2462
Owner_Type_Fourth & Above    1.0209
Owner_Type_Second            1.1939
Owner_Type_Third             1.1507
Brand_audi                 162.4006
Brand_bentley                2.1039
Brand_bmw                  181.4646
Brand_chevrolet             82.7823
Brand_datsun                11.2123
Brand_fiat                  18.2393
Brand_force                  3.0442
Brand_ford                 205.7961
Brand_honda                394.3272
Brand_hyundai              643.9453
Brand_isuzu                  4.0642
Brand_jaguar                33.6676
Brand_jeep                  11.2095
Brand_lamborghini            2.0957
Brand_land-rover            41.5235
Brand_mahindra             178.7714
Brand_maruti               698.0909
Brand_mercedes-benz        215.3314
Brand_mini                  22.3764
Brand_mitsubishi            23.2250
Brand_nissan                65.2061
Brand_porsche               13.4755
Brand_renault              107.3247
Brand_skoda                127.5761
Brand_tata                 129.3071
Brand_toyota               253.3636
Brand_volkswagen           222.3464
Brand_volvo                 13.2587
dtype: float64
```

[59]:
```python
# Retrive Coeff values, p-values and store them in the dataframe
olsmod = pd.DataFrame(ols_model.params, columns = ['coef'])
olsmod['pval'] = ols_model.pvalues
```

[60]:
```python
# 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]
```

[60]:

	coef	pval
Location_Mumbai	-0.0466	0.0321
Brand_datsun	-0.6129	0.0157
Location_Delhi	-0.0584	0.0100
Brand_tata	-0.6380	0.0085
Location_Jaipur	-0.0698	0.0032
Brand_land-rover	0.7524	0.0021
Brand_mini	0.8962	0.0003
Owner_Type_Third	-0.1193	0.0000
Location_Coimbatore	0.0978	0.0000
Fuel_Type_Electric	1.1024	0.0000
Fuel_Type_Diesel	0.2170	0.0000
Location_Hyderabad	0.1224	0.0000
Seats	0.0461	0.0000
Owner_Type_Second	-0.0701	0.0000
Location_Bangalore	0.1664	0.0000
Engine	0.0002	0.0000
Transmission_Manual	-0.1103	0.0000
Mileage	-0.0156	0.0000
Kilometers_driven_log	-0.0668	0.0000
Location_Kolkata	-0.2340	0.0000
const	3.2130	0.0000
Power	0.0051	0.0000
Age	-0.1196	0.0000

[61]:
```
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(y1_train, X1_train)
print(lzip(name, test))
```

```
[('F statistic', np.float64(1.1129073205009117)), ('p-value',
np.float64(0.007683901524057650))]
```

Observations:

We can see that the p-value is less than 0.05, so we can accept the null hypothesis which is the indication that the residuals have **homoscedastic**. And we conclued that Residuals are not **hetroscedastic**

1.8.2 Ridge/Lasso Regression

[62]:
```
# list of alphas to tune
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}

# Applying Lasso
lasso = Lasso()

# cross validation
```

```
folds = 5
model_cv = GridSearchCV(estimator = lasso, param_grid = params, scoring=␣
 ↪'neg_mean_absolute_error',
                        cv = folds, return_train_score=True, verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

[62]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

[63]:
```
# Get the best model and results
print("Best cross-validation score: ", model_cv.best_score_)
print("Best parameters: ", model_cv.best_params_)
```

```
Best cross-validation score:  -0.18051152071846138
Best parameters:  {'alpha': 0.0001}
```

[64]:
```
# Lasso Model for best param
lasso = Lasso(alpha=model_cv.best_params_['alpha'])
lasso.fit(X_train, y_train)
```

[64]:
```
Lasso(alpha=0.0001)
```

[65]:
```
lasso_model_train_perf = model_performance_regression(lasso, X_train, y_train)
lasso_model_train_perf
```

[65]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2381	0.1777	0.9267	0.9258

[66]:
```
lasso_model_test_perf = model_performance_regression(lasso, X_test, y_test)
lasso_model_test_perf
```

[66]:

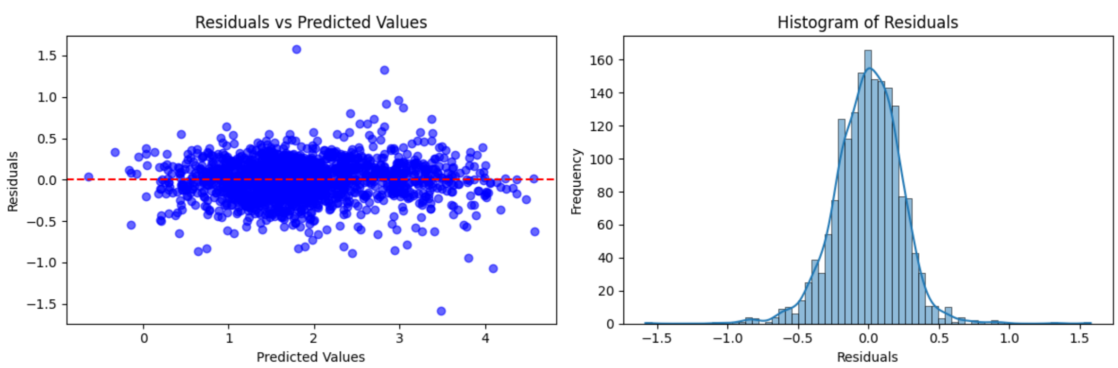
	RMSE	MAE	R-squared	Adj. R2
0	0.2361	0.1777	0.9244	0.9221

Observation:

We can see that Lasso model represent a good scores. Both train set and test set produce almost same scores which is good indication that our model doesn't underfit or overfit.

[67]:
```
# make predictions on the train set
y_train_pred_lasso = lasso.predict(X_train)
y_test_pred_lasso = lasso.predict(X_test)
```

```python
# Calculate residuals
res_lasso = y_test - y_test_pred_lasso

# Scatter plot of residuals vs predicted values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(y_test_pred_lasso, res_lasso, color='blue', alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')

# Histogram of residuals
plt.subplot(1, 2, 2)
sns.histplot(res_lasso, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')

# Show the plots
plt.tight_layout()
plt.show()
```



Observation

- We can see that there is no pattern in the residuals vs fitted values scatter plot and we have our linearity satisfied.
- We can see that the error terms are normally distributed. The assumption of normality is satisfied.

[68]:
```python
ridge_reg = Ridge(alpha=0.1)
ridge_reg.fit(X_train, y_train)
```

```
ridge_model_train_perf = model_performance_regression(ridge_reg, X_train,␣
 ↪y_train)
ridge_model_train_perf
```

[68]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2378	0.1777	0.9269	0.9260

[69]:

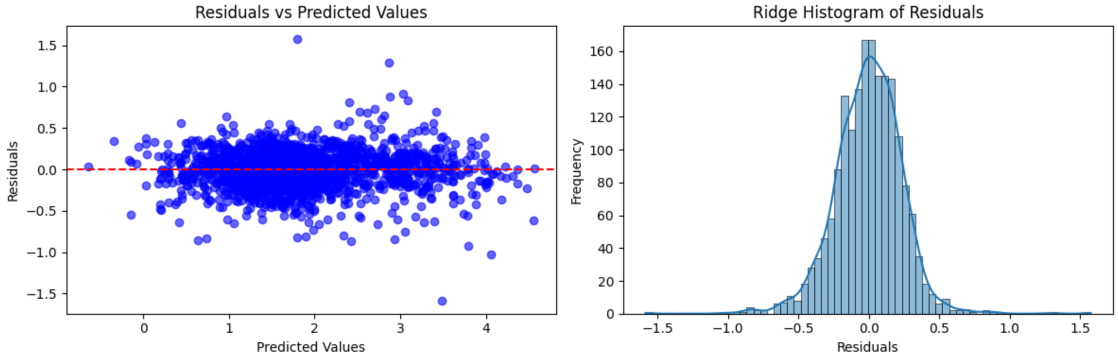
```
ridge_model_test_perf = model_performance_regression(ridge_reg, X_test, y_test)
ridge_model_test_perf
```

[69]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2354	0.1778	0.9248	0.9226

[70]:

```
# make predictions on the train set
y_train_pred_ridge = ridge_reg.predict(X_train)
y_test_pred_ridge = ridge_reg.predict(X_test)

# Calculate residuals
res_ridge = y_test - y_test_pred_ridge

# Scatter plot of residuals vs predicted values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(y_test_pred_ridge, res_ridge, color='blue', alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')

# Histogram of residuals
plt.subplot(1, 2, 2)
sns.histplot(res_ridge, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Ridge Histogram of Residuals')

# Show the plots
plt.tight_layout()
plt.show()
```



Observation

- We can see that there is no pattern in the residuals vs fitted values scatter plot and we have our linearity satisfied.
- We can see that the error terms are normally distributed. The assumption of normality is satisfied.

1.8.3 Decision Tree Regressor

[71]:
```
dtree = DecisionTreeRegressor(random_state=1, max_depth=4)
dtree.fit(X_train, y_train)
```

[71]:
```
DecisionTreeRegressor(max_depth=4, random_state=1)
```

[72]:
```
dtree_model_train_perf = model_performance_regression(dtree, X_train, y_train)
dtree_model_train_perf
```

[72]:

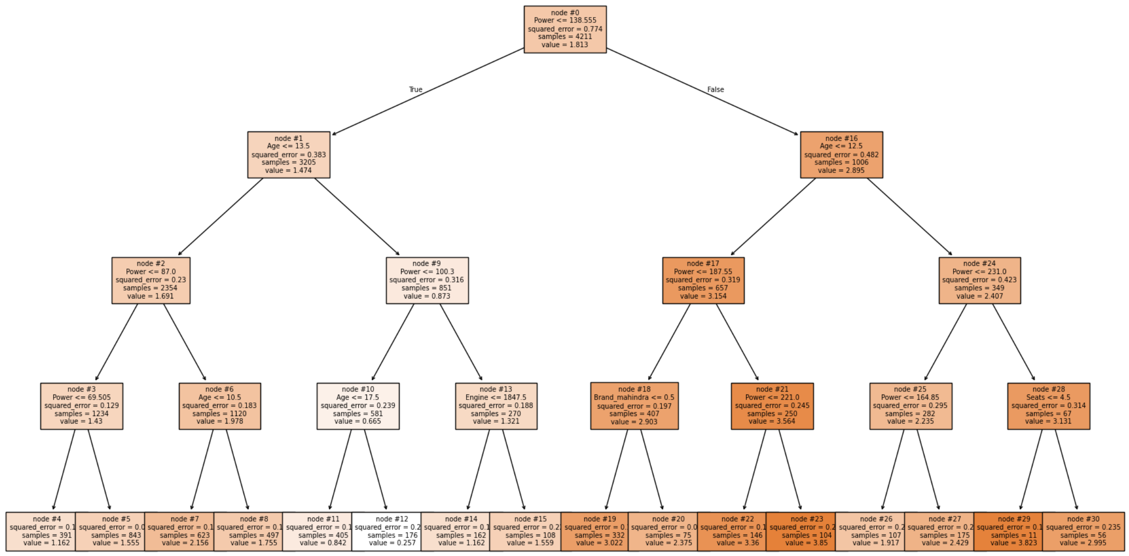
	RMSE	MAE	R-squared	Adj. R2
0	0.3768	0.2822	0.8165	0.8142

[73]:
```
dtree_model_test_perf = model_performance_regression(dtree, X_test, y_test)
dtree_model_test_perf
```

[73]:

	RMSE	MAE	R-squared	Adj. R2
0	0.3742	0.2828	0.8101	0.8044

[74]:
```
plt.figure(figsize = (21, 12))
plot_tree(dtree, feature_names = X_train.columns, filled = True, fontsize = 7,
               node_ids = True, class_names = None)
plt.show()
```



[75]:

```python
print(export_text(dtree, feature_names=X_train.columns.tolist(),
    show_weights=True))
```

```
|--- Power <= 138.56
|   |--- Age <= 13.50
|   |   |--- Power <= 87.00
|   |   |   |--- Power <= 69.51
|   |   |   |   |--- value: [1.16]
|   |   |   |--- Power >  69.51
|   |   |   |   |--- value: [1.55]
|   |   |--- Power >  87.00
|   |   |   |--- Age <= 10.50
|   |   |   |   |--- value: [2.16]
|   |   |   |--- Age >  10.50
|   |   |   |   |--- value: [1.76]
|   |--- Age >  13.50
|   |   |--- Power <= 100.30
|   |   |   |--- Age <= 17.50
|   |   |   |   |--- value: [0.84]
|   |   |   |--- Age >  17.50
|   |   |   |   |--- value: [0.26]
|   |   |--- Power >  100.30
|   |   |   |--- Engine <= 1847.50
|   |   |   |   |--- value: [1.16]
|   |   |   |--- Engine >  1847.50
|   |   |   |   |--- value: [1.56]
|--- Power >  138.56
```

```
|   |--- Age <= 12.50
|   |   |--- Power <= 187.55
|   |   |   |--- Brand_mahindra <= 0.50
|   |   |   |   |--- value: [3.02]
|   |   |   |--- Brand_mahindra >  0.50
|   |   |   |   |--- value: [2.38]
|   |   |--- Power >  187.55
|   |   |   |--- Power <= 221.00
|   |   |   |   |--- value: [3.36]
|   |   |   |--- Power >  221.00
|   |   |   |   |--- value: [3.85]
|   |--- Age >  12.50
|   |   |--- Power <= 231.00
|   |   |   |--- Power <= 164.85
|   |   |   |   |--- value: [1.92]
|   |   |   |--- Power >  164.85
|   |   |   |   |--- value: [2.43]
|   |   |--- Power >  231.00
|   |   |   |--- Seats <= 4.50
|   |   |   |   |--- value: [3.82]
|   |   |   |--- Seats >  4.50
|   |   |   |   |--- value: [2.99]
```

1.8.4 Random Forest Regressor

It provides training multiple Decision Trees on different subsets of the training data with a random subset of the features considered at each split and it makes predictions for new data by averaging the predictions of all the trees.

[76]:
```
# Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators = 100, random_state = 1)

# Fitting the model
rf_reg.fit(X_train, y_train)

# Model Performance on the test data
rf_reg_perf_test = model_performance_regression(rf_reg, X_test, y_test)

rf_reg_perf_test
```

[76]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2071	0.1465	0.9418	0.9401

Models' Performance Comparison

[77]:
```
models_test_comp_df = pd.concat(
    [
```

```
        lreg_model_test_perf.T,
        lasso_model_test_perf.T,
        ridge_model_test_perf.T,
        dtree_model_test_perf.T,
        rf_reg_perf_test.T
    ],
    axis = 1,
)
models_test_comp_df.columns = [
    "Linear Regressor",
    "Lasso Regressor",
    "Ridge Regressor",
    "DecisionTree Regressor",
    "RandomForest regressor"]

print("Test performance comparison:")
models_test_comp_df.T
```

```
Test performance comparison:
```

[77]:

	RMSE	MAE	R-squared	Adj. R2
Linear Regressor	0.2354	0.1778	0.9248	0.9226
Lasso Regressor	0.2361	0.1777	0.9244	0.9221
Ridge Regressor	0.2354	0.1778	0.9248	0.9226
DecisionTree Regressor	0.3742	0.2828	0.8101	0.8044
RandomForest regressor	0.2071	0.1465	0.9418	0.9401

Observations

- Based on the results obtained after comparing all of the models, the Random Forest Regressor is the best-performing model.
- The Random Forest Regressor has the lowest RMSE and MAE, indicating that the average difference between predicted and actual values is the smallest. It also has a higher R-squared and Adjusted R-squared, indicating that the model explains a significant proportion of the variance in the target variable. It also has a low MAPE, indicating that it has a small average percentage error.

1.8.5 Hyperparameter Tuning: Decision Tree

[78]:
```
# Choose the type of estimator
dtree_tuned = DecisionTreeRegressor(random_state=1)

params = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
    }
```

```
# The grid search
dtr_grid = GridSearchCV(dtree_tuned, param_grid=params, cv=5,␣
↪scoring='neg_mean_squared_error')
dtr_grid_obj = dtr_grid.fit(X_train, y_train)

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

# Fit the best algorithm to the data
dtree_tuned.fit(X_train, y_train)

# Model Performance on the test data
dtree_tuned_reg_perf_test = model_performance_regression(dtree_tuned, X_test,␣
↪y_test)

dtree_tuned_reg_perf_test
```

[78]:

```
     RMSE     MAE  R-squared  Adj. R2
0  0.2741  0.1989     0.8981   0.8951
```

Observation:

A good improvement over non tuned hyperparameters model. But still not the best candidate model.

Feature Importance

[79]:

```
dtree_tuned.feature_importances_
```

[79]:

```
array([9.15799113e-03, 2.09149468e-02, 6.64649993e-01, 3.07082705e-03,
       1.06877437e-02, 2.43298131e-01, 1.36311231e-04, 0.00000000e+00,
       4.71665158e-04, 2.92340090e-04, 1.94059541e-03, 1.64292251e-04,
       2.48809012e-04, 3.39867662e-03, 3.77737018e-04, 6.73183667e-04,
       3.66242401e-04, 0.00000000e+00, 0.00000000e+00, 1.75473697e-03,
       2.57538040e-03, 0.00000000e+00, 7.52586435e-04, 1.78632473e-04,
       5.72840370e-04, 0.00000000e+00, 1.95045629e-04, 1.35548407e-03,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.05807168e-04,
       3.98808928e-03, 6.19665053e-04, 0.00000000e+00, 2.55375509e-05,
       0.00000000e+00, 0.00000000e+00, 1.31424824e-03, 8.88276796e-03,
       1.23715598e-03, 4.20120543e-03, 2.36514396e-03, 0.00000000e+00,
       0.00000000e+00, 5.34389973e-04, 0.00000000e+00, 2.60466836e-03,
       5.18055552e-03, 1.60657351e-03, 0.00000000e+00, 0.00000000e+00])
```

[80]:

```
for importance, name in sorted(zip(dtree_tuned.feature_importances_, X_train.
↪columns),reverse=True)[:5]:
    print (name, importance)
```

```
Power 0.664649992789998
Age 0.2432981313313111
```

```
Engine 0.020914946811168575
Kilometers_driven_log 0.010687743718366891
Mileage 0.009157991127588423
```

1.8.6 Hyperparameter Tuning: Random Forest

[81]:
```python
rf_tuned = RandomForestRegressor(random_state = 1)

# Grid of parameters to choose from
rf_parameters = {
    "n_estimators": [100, 110, 120],
    "max_depth": [5, 7, None],
    "max_features": [0.8, 1]
    }

# Run the grid search
rf_grid_obj = GridSearchCV(rf_tuned, rf_parameters, scoring =␣
 ↪'neg_mean_squared_error', cv = 5)

rf_grid_obj = rf_grid_obj.fit(X_train, y_train)

# Set the rf_tuned_regressor to the best combination of parameters
rf_tuned_reg = rf_grid_obj.best_estimator_
rf_tuned_reg.fit(X_train, y_train)

# Model Performance on the test data
rf_tuned_reg_perf_test = model_performance_regression(rf_tuned_reg, X_test,␣
 ↪y_test)

rf_tuned_reg_perf_test
```

[81]:

	RMSE	MAE	R-squared	Adj. R2
0	0.2015	0.1438	0.9449	0.9433

Observation:

Slight improvement over non tuned hyperparameters model

Feature Importance

[82]:
```python
rf_tuned_reg.feature_importances_
```

[82]:
```
array([1.77239296e-02, 1.09591103e-01, 5.31155118e-01, 4.25014842e-03,
       2.10497004e-02, 2.28350951e-01, 1.42504203e-03, 1.04265952e-03,
       2.43165847e-03, 1.26234787e-03, 3.32495114e-03, 1.36371172e-03,
       1.31285585e-03, 4.60854545e-03, 1.49890690e-03, 1.27371833e-03,
       5.22899549e-03, 2.76790879e-04, 8.12803235e-06, 2.01114426e-03,
       2.25926981e-02, 5.84663610e-05, 1.73997690e-03, 7.91698957e-04,
       1.20905648e-03, 3.97213433e-06, 6.09355174e-04, 1.58343592e-03,
```

```
1.75451272e-05, 2.74370735e-04, 5.76500574e-06, 5.73956896e-04,
3.46353819e-03, 1.29490910e-03, 1.40791744e-05, 1.85444837e-04,
1.36525460e-04, 1.96218613e-05, 1.80131746e-03, 2.71928166e-03,
1.24621842e-03, 4.45660484e-03, 3.49862213e-03, 1.75173961e-04,
2.01656441e-04, 8.39201098e-04, 2.28418826e-04, 2.50229615e-03,
4.81280348e-03, 2.37636468e-03, 1.32970024e-03, 4.75180232e-05])
```

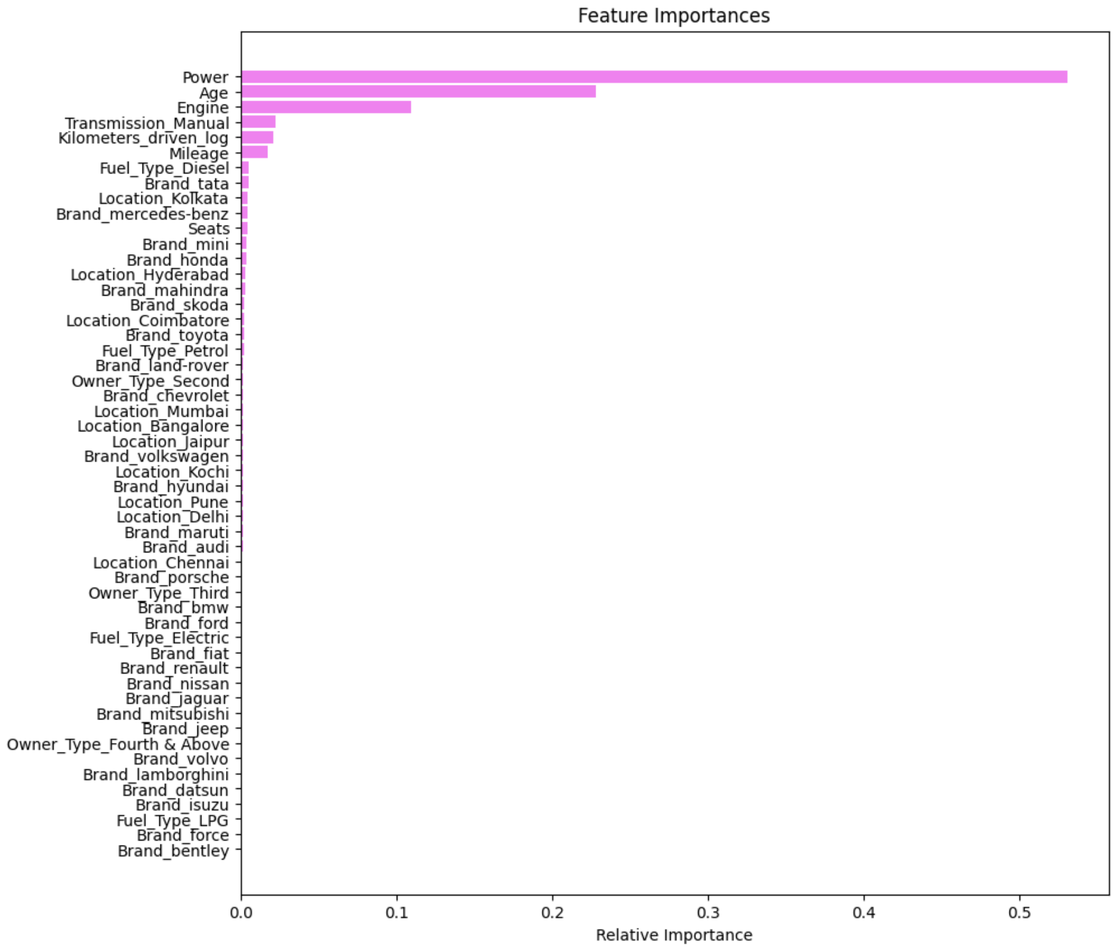
[83]:

```
for importance, name in sorted(zip(rf_tuned_reg.feature_importances_, X_train.
↪columns),reverse=True)[:5]:
    print (name, importance)
```

```
Power 0.531155117787789
Age 0.22835095109577436
Engine 0.10959110296716233
Transmission_Manual 0.02259269805674516
Kilometers_driven_log 0.02104970040521124
```

[84]:

```
feature_names = X_train.columns
importances = rf_tuned_reg.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(10, 10))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet",␣
↪align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Observations:

- We can clear observe that the 5 most important feature are **Power, Age, Engine, Transmission_type and Kilometer_drive**.
- **Power** has the most positive impact on used care price
- **Age** has negative impact on used care price
- **Engine** has positive impact on the used care price but not as important than age and power.
- **Transmission_type** has positive impact on the used care price but lower than above ones.
- Following the above mentioned coeficient we have **Kilometer_driven** and **Mileage** which have both negative affect but much less than 3 first coeficients mentioned.
- **Brands** and **Location** have some positive and negative impact on the prices but they are very low compare to others.

1.9 Conclusions and Recommendations

Comparison of various techniques and their relative performance based on chosen Metric (Measure of success):

[85]:
```python
models_test_comp_df = pd.concat(
    [
        lreg_model_test_perf.T,
        lasso_model_test_perf.T,
        ridge_model_test_perf.T,
        dtree_model_test_perf.T,
        rf_reg_perf_test.T,
        dtree_tuned_reg_perf_test.T,
        rf_tuned_reg_perf_test.T
    ],
    axis = 1,
)
models_test_comp_df.columns = [
    "Linear Regressor",
    "Lasso Regressor",
    "Ridge Regressor",
    "DecisionTree Regressor",
    "RandomForest regressor",
    "DecisionTree Tuned Regressor",
    "RandomForest Tuned Regressor"
    ]

print("Test performance comparison:")
models_test_comp_df.T
```

Test performance comparison:

[85]:

	RMSE	MAE	R-squared	Adj. R2
Linear Regressor	0.2354	0.1778	0.9248	0.9226
Lasso Regressor	0.2361	0.1777	0.9244	0.9221
Ridge Regressor	0.2354	0.1778	0.9248	0.9226
DecisionTree Regressor	0.3742	0.2828	0.8101	0.8044
RandomForest regressor	0.2071	0.1465	0.9418	0.9401
DecisionTree Tuned Regressor	0.2741	0.1989	0.8981	0.8951
RandomForest Tuned Regressor	0.2015	0.1438	0.9449	0.9433

Conclusion:

- Linear Regressor and Lasso regressor have almost the same scores which are good.
- Decision Tree Regressor has the lowest scores even after tunning its hyperparameters. The scores are below Linear and Lasso regressors.
- Random Forest Regressor has the best score in all these models.
- We can observe that after tuning Random Forest hyperparameters we still get slightly better scores.
- Because the Random Forest model performs well on test data, it is not overfitting the training data.
- The Random Forest has a longer runtime in comparison to other models like Decision Tree. Hence, there is a trade-off between runtime and model performance. In this case, we are prioritizing the model performance over runtime, but other approaches are possible depending

on the scenario.

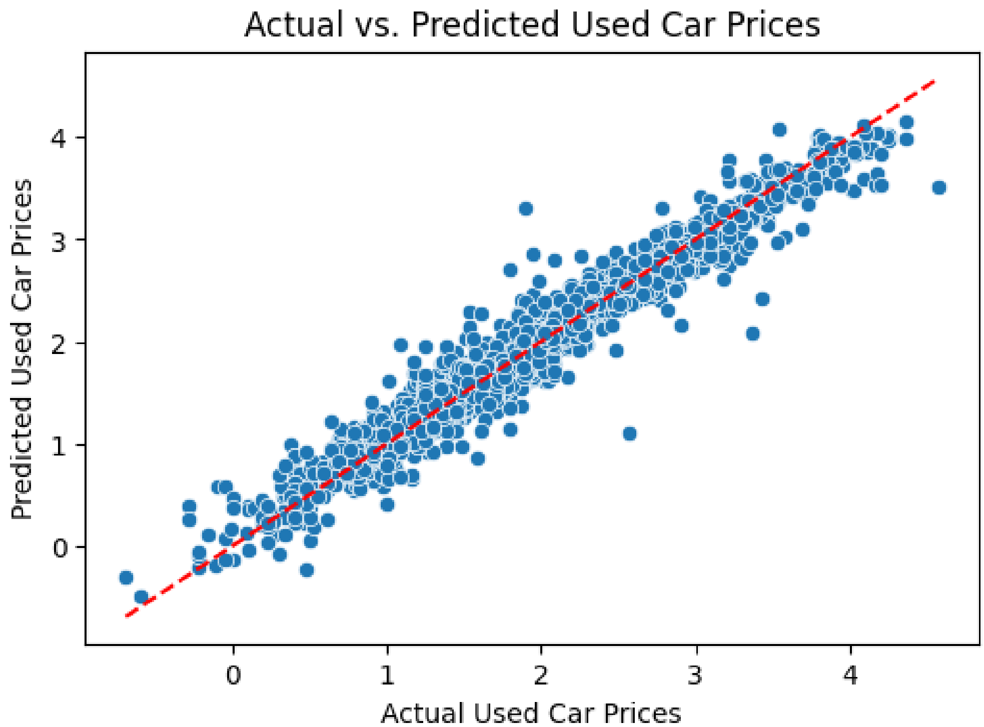
1.9.1 Refined insights:

- We performed EDA, univariate and bivariate analysis, on all the variables in the dataset.
- We checked univariant observations for finding data densities and outlier for all features one by one.
- We check the correlation between features using a heatmap matrix to find more correlated features.
- We studied bivariant data which have the highest correlation number from the heatmap.
- We have treated all missing values of independent variables and we have imputed them mostly by the mean values if the features was mostly normally distributed like **Mileage** and we have imputed others with median when they were skewed to one side.
- We dropped the New_price column because >85% of its values were missing.
- We have dropped the target variable missing values which we can risk to overfit or underfit the models by trying to impute them with medians.
- We started the model building process with all the features trying different model regressors.
- By performing different regressor models and verifying their scores and finding the best fitting model for the used care price prediction.
- Finally, we evaluated the model using different evaluation metrics.

1.9.2 Proposal for the final solution design:

- We have decided to use Random Forest model base on its high scores compare to other models in the regressor model that we have tried in our research.
- We can make a conclusion that Random Forest model has calculated the feature coeficients better than other models.
- In below graph we display how this model acurately predicts the used car prices compare to actual used car prices.

[86]:
```python
# Make predictions
y_pred = rf_tuned_reg.predict(X_test)

# Scatter plot
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test, y=y_pred)

# Reference line
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
 ↪linestyle='--')
plt.xlabel("Actual Used Car Prices")
plt.ylabel("Predicted Used Car Prices")
plt.title("Actual vs. Predicted Used Car Prices")
plt.show()
```



2 Executive Summary

This project developed a machine learning model to accurately predict used car prices. By leveraging a comprehensive dataset of used car attributes, the model achieved high accuracy in estimating market value. The model demonstrates the potential to be a valuable tool for the dealers in the used car market, enabling more informed decision-making.

2.0.1 Problem Review

The used car prices are influenced by numerous factors and constantly changing by economy and market conditions which makes it deficult for car dealers to predict market value of a used car. Based on these factors, the car dealers constantly need to track buyers preferences and market trends fluctuations influencing buyers demands.

2.0.2 Purposed Solution

Our solution to address this problem of predicting the price of a car, it is to develop a machine learning model which can accurately predict the selling price of an used car. The model is build on a provided hitorical used car sales data which include information factors such as:

- **Cars conditions:** Age, Mileage, Engine, Power, Kilometers_driven.
- **Buyers prefernece s:** Make/Brand, fuel type, number of seats, and the owner type (sold by first owner or higher).

The model leverages data on used car attributes to build a predictive regression model, which can then be used to estimate the market value of specific used cars.

In used car price prediction regression models, several factors consistently emerge as highly influential. Car age, power, engine, transmission type and kilometers_driven are among the most significant predictors, with age having the strongest negative impact and Power with highest positive impact on pricing. Other important factors include fuel type, number of previous owners, owner type and number of seats have impact on sales number.

2.0.3 Insight about influenced factors

The following insights highlight the importance of considering a combination of factors when predicting used car prices, rather than relying on any single factor alone. Understanding their relationships helps in developing more accurate prediction models and making informed decisions for dealers.

Age:

The age of a car has a substantial negative impact on its price, as older cars typically depreciate more rapidly and usually have more mileage.

Power:

A car's power can influence its price positively. More powerful engines may be associated with higher-end models and better performance, leading to higher prices.

Mileage and Kilometers_driven:

The total distance a car has been driven is a strong indicator of its wear and tear and, consequently, its price. These 2 factors have negative influence on the car price.

Engine:

Engine is another infuencial factor on the car price. Bigger engine has more power and has a positive influence on the used car price.

Fuel Type:

The type of fuel a car uses can affect its price, diesel and petrol fuel type cars are more sold cars over electric and other types. Electic cars may don't have enough ducking electrical stations yet available. Even if the diesel vehicule prices are as high as electical types. That can be explained by their popularity.

Transmission Type:

The most sold cars are manual transmission while the automatic transmissions have the highest sold price which can explain why manual transmissions are more popular. So price here can be a determining factor when it comes to select a manual car over automatic.

Brand:

Certain car brands are perceived as more reliable or desirable, leading to higher solded cars. In indian market Maruti and Hyndai have the most sold brands followed by Honda and Toyota.

Seller Owner Type:

Whether the car is sold has only owned by first owner or owned by more than one can influence sales number because they first owner cars are sold more. Cars with fewer previous owners are generally perceived as being in better condition, potentially leading to higher prices.

Car Seats:

Car seats can infuence the car price as by demand or by model. It means car with 4 or 5 seats are more popular and their price is higher. 2 seated cars are mostly expensive sports cars and 7 seats cars usually are larger SUV type cars.

Location:

Geographical location factors can give us insight about the bigger market where more buyers are competing for buying a good used car. Analysing the data gives us insight about the best markets for selling car which mostly are the big cities with higher population and wealth.

2.0.4 Prediction Model Evaluation:

For model evaluation, by comparing models accuracy metrics. Condsidering appropriete metrics like R-squared, Mean Absolute Error (MAE), and Mean Squared Error (MSE) to assess model performance and identifying areas for improvement. Considering improving model performance with more feature engineering by avoiding overfitting and improve model interpretability.

2.0.5 Prediction Model Selection:

The selection of the model is based on the best model accuracy. Linear regression has a interpretation features for get sense of influencing factors in building regression models. However, it is recommended to consider more complex models like **Random Forest** for potentially better accuracy with R-squard equal to 94.5%.

2.0.6 Data Processing and Feature Selection

Handling missing data, outlier detection, and feature engineering are crucial for building robust models, trying to remove features that have no significant influence on prices. Creating new feature from existing features that can have impact on the prices. Avoiding overfitting and improve model interpretability by making appropriete feature selection.

2.1 Recommendation:

As we have selected our model regressor for our prediction model. Therefor, based on this model and its outputs, several insights can be drawn to aid stakeholders in understanding the factors that significantly influence car prices in the dataset. Notably, the model involving Power and Engine have the highest positive influence on car price and **Age** and **Kilometers_driven** have negative influence on the price. By effectively communicating these features, stakeholders can better develop pricing strategies.

Car models with significant categorical predictors such as **Transmission type**, **Fuel_Type** and **number of seats** provide actionable insights, where **manual** transmission impacts positively the sale, **diesel** fuel type and 4/5 seats cars are considered as popular and add value to cars, which stakeholders can consider for using as targeted marketing and pricing strategies.

Brands can be considered for forcasting car inventories. Car dealers can consider higher purcentage car inventory of Maruti, Hyundai and Honda and smaller purcentage of sports and luxury cars.

2.2 Implementation:

We have tried multiple regression models and by comparing their outputs, **Random Forest** proved superior in terms of handling multicollinearity, which is common with the extensive variables in our dataset. The model has an R-squared value of 0.945, indicating strong explanatory power. On the other hand, linear and Ridge regression, by incorporating regularization, offered also good preformance, confirming their suitability for predicting used car prices in scenarios with complex and interrelated predictors.

2.3 In conclusion:

our technical analysis is based on comparing multiple regression techniques which incorporate the impact of various car attributes on pricing. Our selected model shows with high degree of confidence and small slight difference between actual price and predicted price that it can be applied practically for pricing strategies in the used car market. While, this approach ensures that stakeholders can make informed decisions based on robust data-driven insights. The used car price can be influence a lot by other factors which are missing in this used historical data, such as a car's body condition not included our model for prediction. This is something that we need to include in future in our model. On the other hand, the historical data used for building our model is 5 years old and we need to tune our model with newest data for getting better insight about influencing factors. They can change over time with the changes in social economic factors.