FLIP ROBO

Car Price Prediction Project

Submitted by:

SONALI WARKHADE

ACKNOWLEDGMENT

I would like to express my thanks of gratitude to mentors Shubham Yadav as well as Flip Robo who gave me the golden opportunity to do this wonderful project on the topic Micro Credit Defaulter, which also helped me in doing a lot of research and I came to know about so many new things. I am really very thankful to them.

I am making this project to increase my knowledge.

INTRODUCTION

Problem Statement:

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model.

This project consists on of 2 phases:

Data Collection Phase

We need to scrape at least 5000 used cars data.

In this section we need to scrape the data of used cars from websites. We need web scraping for this. we have to fetch data for different locations. The number of columns for data doesn't have limit. Generally, these columns are Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location and at last target variable Price of the car. This data is to give you a hint about important variables in used car model.

Model Building Phase

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model.

Analytical Problem Framing

With the help of Selenium I have scrapped data from different websites like cartrade, Olx and carwale and used Pandas library to save the data in excel file. Just taking a glace on basic code for scrapping from different websites.

```
# Opening the homepage of cartrade
#delhi
url = "https://www.cartrade.com/"
driver.get(url)

time.sleep(2)
driver.find_element_by_xpath('//*[@id="ucity"]').click()
driver.find_element_by_xpath('//*[@id="ucity"]/optgroup[1]/option[1]').click()
driver.find_element_by_xpath('//*[@id="rvwtop"]/div/div[1]/div[2]/div[2]/input').click()
time.sleep(2)
bodytype = driver.find_element_by_xpath('//*[@id="selectlistarrow6"]')
Hatchback = driver.find_element_by_xpath('//*[@id="body_Hatchback"]')
Sedan = driver.find_element_by_xpath('//*[@id="body_Sedan"]')
SUV = driver.find_element_by_xpath('//*[@id="body_SUV"]')
Van_Minivan = driver.find_element_by_xpath('//*[@id="body_Van_Minivan"]')
action= ActionChains(driver)
action.move_to_element(bodytype).move_to_element(Hatchback).click().move_to_element(Sedan).click() \
.move_to_element(SUV).click().move_to_element(Van_Minivan).click().perform()
```

```
links1=[]
driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
time.sleep(2)
try:
    for l in driver.find_elements_by_xpath('//h2[@class="h2heading truncate"]/a'):
        links1.append(l.get_attribute('href'))
        time.sleep(1)
except NoSuchElementException:
        links1.append("-")
        time.sleep(1)
for page in range(0,10):
        nxt_button=driver.find_elements_by_xpath('//div[@class="pagination"]/ul/li[@class="next"]/a')#scraping the list of b
        time.sleep(2)
        try:
            driver.get(nxt_button[1].get_attribute('href'))#getting the link from the list for next page
            time.sleep(2)
        except:
            driver.get(nxt_button[0].get_attribute('href'))
            time.sleep(2)
        driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
        time.sleep(2)
        try:
            for l in driver.find_elements_by_xpath('//h2[@class="h2heading truncate"]/a'):
                links1.append(l.get_attribute('href'))
                time.sleep(1)
        except NoSuchElementException:
                links1.append("-")
                time.sleep(1)
```

```
for url in links:
    driver.get(url)
    time.sleep(2)

    try:
        # Extracting Brand
        try:
            b = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[1]/div/ul/li[4]/a/span')
            Brand.append(b.text)
        except NoSuchElementException:
            Brand.append('-')

        time.sleep(1)
        # Extracting Model
        try:
            m = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[1]/div/ul/li[5]/a/span')
            Model.append(m.text)
        except NoSuchElementException:
            Model.append('-')

        time.sleep(1)
        # Extracting Variant
        try:
            v = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[1]/div/ul/li[6]/span')
            Variant.append(v.text)
        except NoSuchElementException:
            Variant.append('-')
        # Extracting Man_year
        try:
            my = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div[1]/div[4]/table/tbody/tr
            Man_year.append(my.text)
        except NoSuchElementException:
            Man_year.append('-')
        # Extracting Driven_km
        try:
            dk = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div[1]/div[4]/table/tbody/tr
            Driven_km.append(dk.text)
        except NoSuchElementException:
            Driven_km.append('-')
        # Extracting Fuel
        try:
            f = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div[1]/div[4]/table/tbody/tr[
            Fuel.append(f.text)
        except NoSuchElementException:
            Fuel.append('-')
        # Extracting Num_of_owners
        try:
            no = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div[1]/div[4]/table/tbody/tr
            Num_of_owners.append(no.text)
        except NoSuchElementException:
            Num_of_owners.append('-')
        # Extracting Location
        try:
            l = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div/div[4]/table/tbody/tr[1]/
            Location.append(l.text)
        except NoSuchElementException:
            Location.append('-')
        # Extracting Price
        try:
            p = driver.find_element_by_xpath('//*[@id="idbybody"]/div[2]/div[8]/div[1]/div[2]/div/div[1]/div[1]/span[2]')
            Price.append(p.text)
        except NoSuchElementException:
            Price.append('-')
    except TimeoutException:
        pass
    except NoSuchElementException:
        pass
```

```
# dictionary of lists
dict1 = {'Brand': Brand, 'Model': Model, 'Variant': Variant,'Man_year':Man_year,'Driven_km':Driven_km,'Fuel':Fuel,
         'Num_of_owners':Num_of_owners,'Location':Location,'Price':Price}

df = pd.DataFrame(dict1)

# saving the dataframe
df.to_csv('cartrade.csv')
```

```
links1=[]
for page in range(0,7):
    try:
        driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
        driver.find_element_by_xpath('//li[@class="_20FqS"]/div[@class="JbJAl"]').click()
        time.sleep(1)
    except NoSuchElementException:
        pass
```

```
try:
    for l in driver.find_elements_by_xpath('//*[@id="container"]/main/div/div/section/div/div/div[5]/div[2]/div/div[2]/ul/l
        links1.append(l.get_attribute('href'))
        time.sleep(1)
except NoSuchElementException:
        links1.append("-")
        time.sleep(1)
```

```
len(links1)
```

320

```
Brand=[]
Model=[]
Variant=[]
Man_year=[]
Driven_km=[]
Fuel=[]
Num_of_owners=[]
Location=[]
Price=[]
```

```
for url in links1:
    driver.get(url)
    time.sleep(2)

    try:
        # Extracting Brand
        try:
            b = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/di
            Brand.append(b.text)
        except NoSuchElementException:
            Brand.append('-')

        time.sleep(1)
        # Extracting Model
        try:
            m = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/di
            Model.append(m.text)
        except NoSuchElementException:
            Model.append('-')

        time.sleep(1)
        # Extracting Variant
        try:
            v = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/di
            Variant.append(v.text)
        except NoSuchElementException:
            Variant.append('-')
        # Extracting Man_year
        try:
            my = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/d:
            Man_year.append(my.text)
        except NoSuchElementException:
            Man_year.append('-')
        # Extracting Driven_km
        try:
            dk = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/d:
            Driven_km.append(dk.text)
        except NoSuchElementException:
            Driven_km.append('-')
        # Extracting Fuel
        try:
            f = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/di
            Fuel.append(f.text)
        except NoSuchElementException:
            Fuel.append('-')
        # Extracting Num_of_owners
        try:
            no = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[4]/section[1]/div/div/div[1]/div/d:
            Num_of_owners.append(no.text)
        except NoSuchElementException:
            Num_of_owners.append('-')
        # Extracting Location
        try:
            l = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[5]/div[1]/div/section/div/div[1]/di
            Location.append(l.text)
        except NoSuchElementException:
            Location.append('-')
        # Extracting Price
        try:
            p = driver.find_element_by_xpath('//*[@id="container"]/main/div/div/div/div[5]/div[1]/div/section/span[1]')
            Price.append(p.text)
        except NoSuchElementException:
            Price.append('-')
    except TimeoutException:
        pass
    except NoSuchElementException:
        pass
```

```
# dictionary of lists
dict2 = {'Brand': Brand, 'Model': Model, 'Variant': Variant,'Man_year':Man_year,'Driven_km':Driven_km,'Fuel':Fuel,
         'Num_of_owners':Num_of_owners,'Location':Location,'Price':Price}

df1 = pd.DataFrame(dict2)

# saving the dataframe
df1.to_csv('Olx.csv')
```

```
# Opening the homepage of cartrade
#Delhi
url = "https://www.carwale.com/used/cars-for-sale/#sc=-1&so=-1&pn=1"
driver.get(url)

time.sleep(2)
driver.find_element_by_xpath('//*[@id="closeLocIcon"]').click()
time.sleep(2)

driver.find_element_by_xpath('//*[@id="drpCity"]').click()
driver.find_element_by_xpath('//*[@id="drpCity"]/option[3]').click()

driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
```

```
links1=[]
driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
time.sleep(1)
driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
time.sleep(1)
driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")
time.sleep(2)
for i in range(1,7):
    try:
        for l in driver.find_elements_by_xpath(f'//*[@id="listing{i}"]/li/div/div/div[1]/a'):
            links1.append(l.get_attribute('href'))
            time.sleep(1)
    except NoSuchElementException:
            pass
            time.sleep(1)
```

```
len(links1)
```

128

```
Brand=[]
Model=[]
Variant=[]
Man_year=[]
Driven_km=[]
Fuel=[]
Num_of_owners=[]
Location=[]
Price=[]
```

```python
for url in links1:
    driver.get(url)
    time.sleep(2)

    try:
        # Extracting Brand
        try:
            b = driver.find_element_by_xpath('/html/body/div[14]/section[2]/div[1]/div[1]/div[1]/ul/li[4]/a/span')
            Brand.append(b.text)
        except NoSuchElementException:
            Brand.append('-')

        time.sleep(1)
        # Extracting Model
        try:
            m = driver.find_element_by_xpath('/html/body/div[14]/section[2]/div[1]/div[1]/div[1]/ul/li[5]/a/span')
            Model.append(m.text)
        except NoSuchElementException:
            Model.append('-')

        time.sleep(1)
        # Extracting Variant
        try:
            v = driver.find_element_by_xpath('/html/body/div[14]/section[2]/div[1]/div[1]/div[1]/ul/li[6]/span[2]')
            Variant.append(v.text)
        except NoSuchElementException:
            Variant.append('-')
        # Extracting Man_year
        try:
            my = driver.find_element_by_xpath('//*[@id="overview"]/div/ul/li[2]/div[2]')
            Man_year.append(my.text)
        except NoSuchElementException:
            Man_year.append('-')
        # Extracting Driven_km
        try:
            dk = driver.find_element_by_xpath('//*[@id="overview"]/div/ul/li[3]/div[2]')
            Driven_km.append(dk.text)
        except NoSuchElementException:
            Driven_km.append('-')
        # Extracting Fuel
        try:
            f = driver.find_element_by_xpath('//*[@id="overview"]/div/ul/li[4]/div[2]')
            Fuel.append(f.text)
        except NoSuchElementException:
            Fuel.append('-')
        # Extracting Num_of_owners
        try:
            no = driver.find_element_by_xpath('//*[@id="overview"]/div/ul/li[9]/div[2]')
            Num_of_owners.append(no.text)
        except NoSuchElementException:
            Num_of_owners.append('-')
        # Extracting Location
        try:
            l = driver.find_element_by_xpath('/html/body/div[14]/section[2]/div[2]/div[1]/div[2]/ul/li[5]/span[2]')
            Location.append(l.text)
        except NoSuchElementException:
            Location.append('-')
        # Extracting Price
        try:
            p = driver.find_element_by_xpath('/html/body/div[14]/section[2]/div[2]/div[2]/div/div[1]/div[1]/div/div[1]/p')
            Price.append(p.text)
        except NoSuchElementException:
            Price.append('-')
    except TimeoutException:
        pass
    except NoSuchElementException:
        pass
```

```python
# dictionary of lists
dict3 = {'Brand': Brand, 'Model': Model, 'Variant': Variant,'Man_year':Man_year,'Driven_km':Driven_km,'Fuel':Fuel,
        'Num_of_owners':Num_of_owners,'Location':Location,'Price':Price}

df3 = pd.DataFrame(dict3)

# saving the dataframe
df3.to_csv('carwale.csv')
df3
```

	Brand	Model	Variant	Man_year	Driven_km	Fuel	Num_of_owners	Location	Price
0	Ford	Figo	Duratorq Diesel LXI 1.4	Jun 2012	70,000 km	Diesel	First	Noida	₹ 1.5 Lakh
1	Mahindra	XUV500	W8	Jun 2012	75,000 km	Diesel	First	Noida	₹ 4 Lakh
2	Mercedes-Benz	E-Class	250 D (W210)	Jun 2013	90,000 km	Diesel	First	Chandigarh	₹ 17.5 Lakh
3	Mercedes-Benz	C-Class	C 220d Progressive [2018-2019]	Jun 2019	23,400 km	Diesel	First	Delhi	₹ 41 Lakh
4	Mahindra	Scorpio	S6 Plus	Mar 2015	75,000 km	Diesel	Second	Delhi	₹ 8.15 Lakh
...	...	...	...	...	...	...	...	...	...
1499	Maruti Suzuki	Baleno	Zeta 1.2 AT	Jun 2020	9,771 km	Petrol	First	Hyderabad	₹ 9.07 Lakh
1500	Hyundai	Santro	GLS	Jun 2012	65,000 km	Petrol	First	Hyderabad	₹ 2.95 Lakh
1501	Maruti Suzuki	Swift	ZDi	Jun 2014	79,000 km	Diesel	First	Hyderabad	₹ 5.95 Lakh
1502	Mahindra	Scorpio	VLX 4WD BS-IV	Jun 2013	72,000 km	Diesel	First	Hyderabad	₹ 7.9 Lakh
1503	Toyota	Fortuner	2.8 4x2 MT [2016-2020]	Jun 2017	1,30,000 km	Diesel	First	Hyderabad	₹ 29.8 Lakh

1504 rows × 9 columns

```python
# dictionary of lists
dict4 = {'Brand': Brand, 'Model': Model, 'Variant': Variant,'Man_year':Man_year,'Driven_km':Driven_km,'Fuel':Fuel,
        'Num_of_owners':Num_of_owners,'Location':Location,'Price':Price}

df4 = pd.DataFrame(dict4)

# saving the dataframe
df4.to_csv('cartrade01.csv')
```

```python
filename=['cartrade.csv', 'Olx.csv','carwale.csv','cartrade01.csv']
```

```python
combined_csv = pd.concat([pd.read_csv(f) for f in filename])
```

```python
combined_csv.to_excel( "Car_Price.xlsx", index=False, encoding='utf-8')
```

```python
combined_csv.to_csv( "CarPrice.csv", index=False, encoding='utf-8')
```

Firstly, we will start by importing required libraries and databases.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression,ElasticNet
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from scipy import stats
from scipy.stats import skew
import pylab


from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

import joblib
import warnings
warnings.filterwarnings('ignore')
```

```
#pd.set_option('display.max_colwidth',100 )
cd=pd.read_excel("Car_Price.xlsx")
data=pd.DataFrame(data=cd)
data
```

	Unnamed: 0	Brand	Model	Variant	Man_year	Driven_km	Fuel	Num_of_owners	Location	Price
0	0	Mercedes-Benz	GLS	350 d	2017	31,000 Kms	Diesel	First	Delhi	62.5 Lakh
1	1	Hyundai	i10	Magna	2008	75,000 Kms	Petrol	First	Noida	1.4 Lakh
2	2	Hyundai	Elantra	1.6 S MT	2013	76,000 Kms	Diesel	First	Delhi	5.25 Lakh
3	3	Toyota	Land Cruiser	LC 200 VX	2019	24,000 Kms	Diesel	First	Delhi	1.65 Crore
4	4	Maruti Suzuki	Ignis	Zeta 1.2 MT	2018	28,000 Kms	Petrol	First	Delhi	4.85 Lakh
...	...	...	...	...	...	...	...	...	...	...
5068	808	Toyota	Fortuner	3.0 MT	2012	60,000 Kms	Diesel	First	Kolkata	12.9 Lakh
5069	809	Honda	WR-V	VX MT Diesel	2017	30,000 Kms	Diesel	First	Kolkata	7.25 Lakh
5070	810	Toyota	Innova	2.5 V 7 STR	2011	70,000 Kms	Diesel	First	Kolkata	5.7 Lakh
5071	811	BMW	X3	xDrive20d	2012	42,000 Kms	Diesel	First	Kolkata	14.5 Lakh
5072	812	Mercedes-Benz	M-Class	ML 250 CDI	2015	36,000 Kms	Diesel	First	Kolkata	32.5 Lakh

5073 rows × 10 columns

```
data.columns
```

```
Index(['Unnamed: 0', 'Brand', 'Model', 'Variant', 'Man_year', 'Driven_km',
       'Fuel', 'Num_of_owners', 'Location', 'Price'],
      dtype='object')
```

Above is the list of all columns.

Dataset has 5073 rows and 10 columns.

```
data.dtypes
```

```
Unnamed: 0        int64
Brand            object
Model            object
Variant          object
Man_year         object
Driven_km        object
Fuel             object
Num_of_owners    object
Location         object
Price            object
dtype: object
```

We can see both type of columns numerical and object type.

```
#pd.set_option('display.max_rows',None)
data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Unnamed: 0	5073	NaN	NaN	NaN	724.608	499.797	0	317	634	1088	1941
Brand	5073	47	Maruti Suzuki	1153	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Model	5073	327	Swift	180	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Variant	5073	1818	Others	151	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Man_year	5073	214	2017	225	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Driven_km	5073	1841	-	178	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Fuel	5073	12	Diesel	2473	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Num_of_owners	5073	11	First	2611	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Location	5073	625	Pune	520	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Price	5073	1356	-	86	NaN	NaN	NaN	NaN	NaN	NaN	NaN

We can see - in some variables most frequently value. We will work on it.

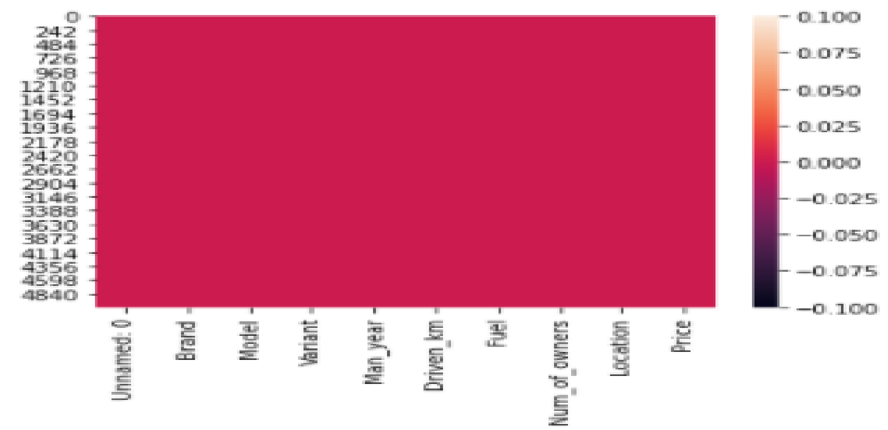
We can see count is all same for all variable, let's check for null values now:

```
data.isnull().sum()
```

```
Unnamed: 0       0
Brand            0
Model            0
Variant          0
Man_year         0
Driven_km        0
Fuel             0
Num_of_owners    0
Location         0
Price            0
dtype: int64
```

```
#Let's see it with the help of heatmap:
sns.heatmap(data.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1764ac752e0>
```



Our dataset does not contain null values.

```
# dropping 'Unnamed: 0 varible as its not further analysis
data.drop('Unnamed: 0', axis=1, inplace=True)
```

```
data['Driven_km'] = data['Driven_km'].str.replace('-', 'NA') # replaced - with NA
data['Driven_km'] = data['Driven_km'].str.replace(' Kms', '') # removed kms as column as column name already has km it in
```

```
data['Price'] = data['Price'].str.replace('₹', '')#removed ₹
data['Price'] = data['Price'].str.replace(',', '')#removed , as well
```

```
#dropping all rows where price column does not contain any price
data = data[~data.Price.str.contains("-") == True]
```

```
def isfloat(value):
    try:
        float(value)
        return float(value)
    except ValueError:
        return value
```

```
data['Price']=data['Price'].apply(isfloat)
```

```
def value_to_float(x):
    if type(x) == float or type(x) == int:
        return x
    if x.isdigit():
        x=float(x)
        return float(x)
    if 'Lakh' in x:
        if len(x) > 1:
            return float(x.replace(' Lakh','')) *100000
        return 100000.0
    if 'Crore' in x:
        if len(x) > 1:
            return float(x.replace(' Crore','')) * 10000000
        return 10000000.0
```

```
#changing all values of price column in numerical type
data['Price']=data['Price'].apply(value_to_float)
```

```
data['Price']
```

```
0        6250000.0
1         140000.0
2         525000.0
3       16500000.0
4         485000.0
           ...
5068     1290000.0
5069      725000.0
5070      570000.0
5071     1450000.0
5072     3250000.0
Name: Price, Length: 4987, dtype: float64
```

```
#checking unique values for all object type columns
for i in data.columns:
    if data[i].dtype == 'object':
        print(i, ":", data[i].nunique())
```

```
Brand : 46
Model : 326
Variant : 1818
Man_year : 213
Driven_km : 1841
Fuel : 12
Num_of_owners : 11
Location : 624
```

we can see manufacturing year has 213, vehicle usually does not last for 200 years. Let’s view it:

```
data['Man_year'].value_counts()
```

```
2017        225
2018        222
2016        196
2015        176
2014        174
           ...
1901          1
Sep 2009      1
Sep 2019      1
Nov 2014      1
1996          1
Name: Man_year, Length: 213, dtype: int64
```

some values contain month as well. Let's remove months and only keep year.

```
data['Man_year'].replace(regex=True,inplace=True,to_replace=r'\D',value=r'')
data['Man_year'].value_counts()
```

```
2017    423
2018    419
2016    392
2014    331
2015    329
       ...
1986      1
1988      1
1989      1
1995      1
1996      1
Name: Man_year, Length: 70, dtype: int64
```

```
#creating cat col list for unique value count till 70 as we will have clear visual for those many values.
cat_col=[]
for i in data:
    if data[i].nunique() <= 70:
        cat_col.append(i)

print(cat_col)
```

```
['Brand', 'Man_year', 'Fuel', 'Num_of_owners']
```

```
# let check null values and value counts for all categorical variables
for i in cat_col:
    print(i,"Column value counts:\n", data[i].value_counts(), "\n")
```

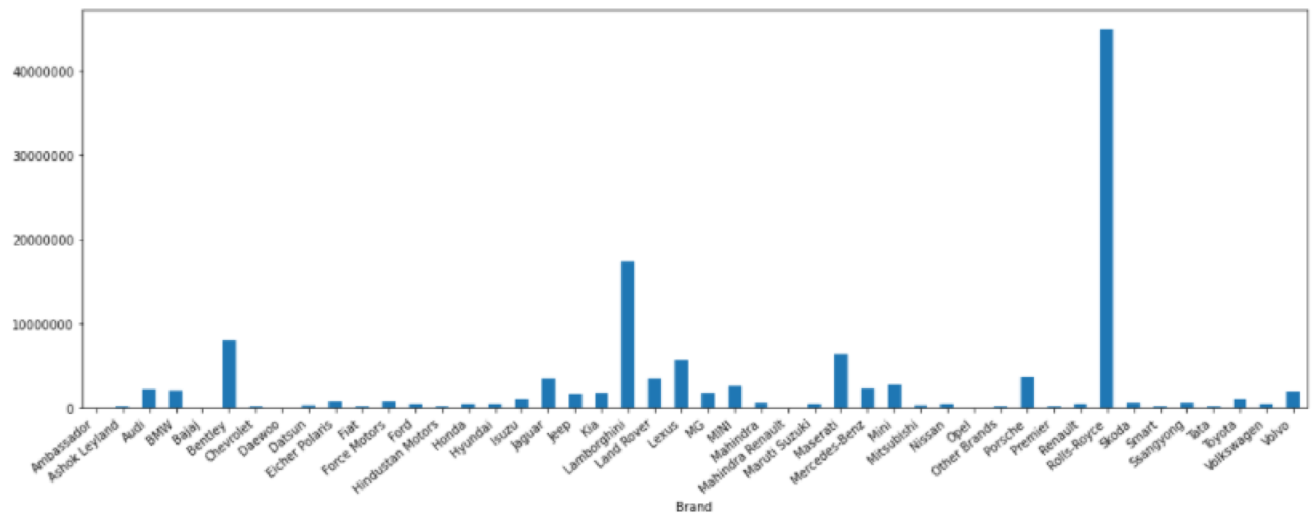
```
Brand Column value counts:
 Maruti Suzuki      1153
Hyundai             779
Honda               411
Toyota              388
Mahindra            291
Mercedes-Benz       280
BMW                 216
Ford                211
Volkswagen          185
Audi                176
Tata                160
Renault             148
Skoda               101
Nissan               80
Chevrolet            58
Land Rover           52
Jaguar               35
Jeep                 29
Datsun               28
MG                   27
Kia                  27
Volvo                26
Fiat                 22
Porsche              17
Other Brands         15
Mitsubishi           14
MINI                  8
Force Motors          8
Mini                  5
Bajaj                 4
Lexus                 4
Bentley               4
Ambassador            3
Ssangyong             3
Isuzu                 3
Ashok Leyland         3
Lamborghini           2
Eicher Polaris        2
Opel                  2
Smart                 1
Rolls-Royce           1
Hindustan Motors      1
Mahindra Renault      1
Maserati              1
Premier               1
Daewoo                1
Name: Brand, dtype: int64
```

```
Man_year Column value counts:
 2017    423
2018    419
2016    392
2014    331
2015    329
       ...
1986      1
1988      1
1989      1
1995      1
1996      1
Name: Man_year, Length: 70, dtype: int64

Fuel Column value counts:
 Diesel            2473
Petrol            2295
CNG & Hybrids       82
-                   57
LPG                 30
CNG                 14
CNG + Cng           14
Petrol + Cng         9
Automatic            4
Electric             4
Hybrid               3
Manual               2
Name: Fuel, dtype: int64

Num_of_owners Column value counts:
 First                 2611
1st                   1008
2nd                    531
Second                 469
3rd                    150
-                      130
Third                   39
4th                     28
4+                      10
UnRegistered Car         9
Fourth                   2
Name: Num_of_owners, dtype: int64
```

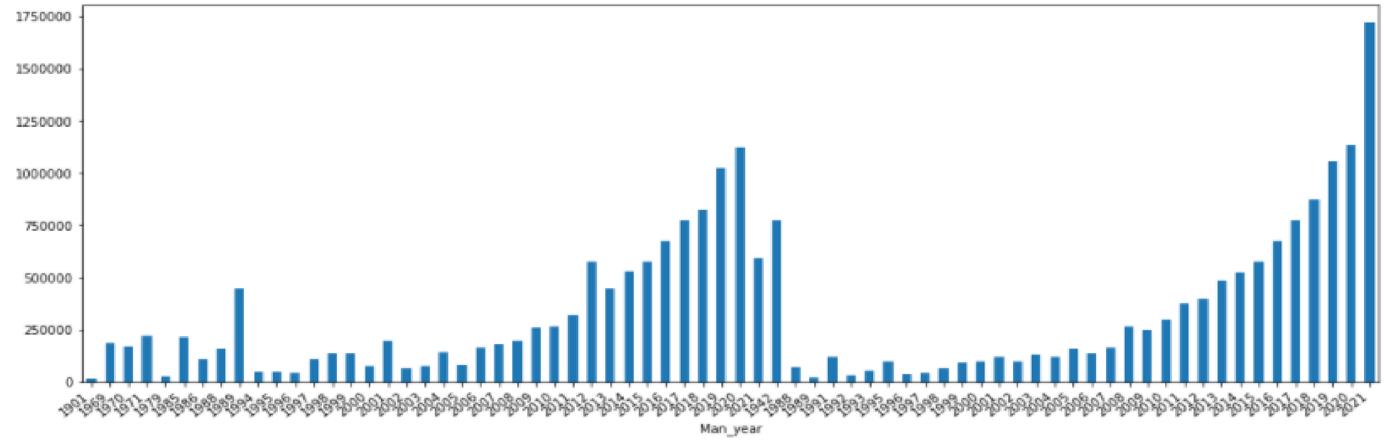
Fuel variable has - value in it. And Number of owners has same value with different format. let’s make required changes.

```
for i in cat_col:
    plt.figure(figsize=(18,6))
    #a=sns.countplot(train[i])
    a=data.groupby(i)['Price'].median().plot.bar()
    print(data[i].value_counts())
    a.set_xticklabels(a.get_xticklabels(), rotation=40, ha="right")
    #a.ticklabel_format(useOffset=False, style='plain')
    plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
    plt.show()
```

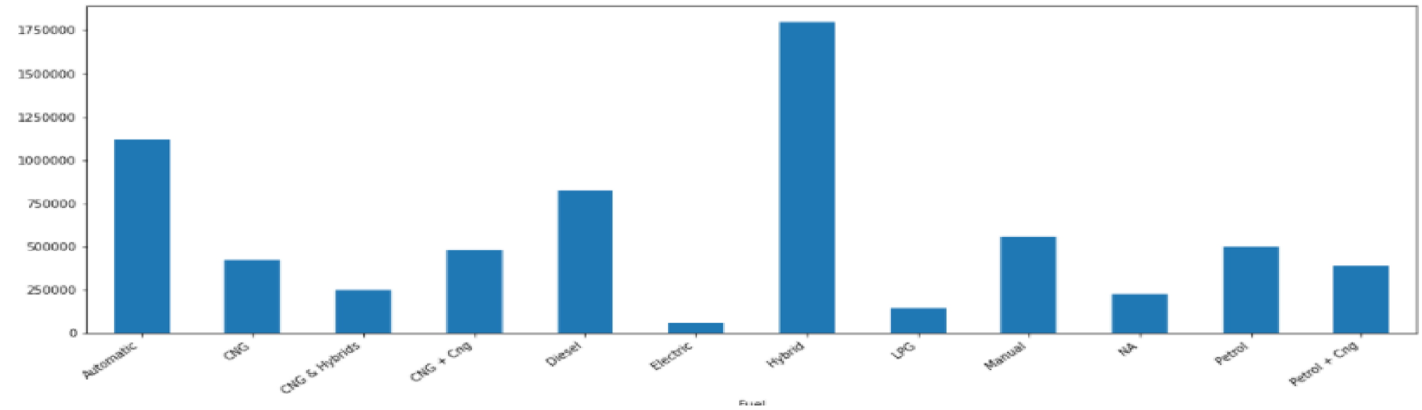
```
Maruti Suzuki       1153
Hyundai              779
Honda                411
Toyota               388
Mahindra             291
Mercedes-Benz        280
BMW                  216
Ford                 211
Volkswagen           185
Audi                 176
Tata                 160
Renault              148
Skoda                101
Nissan                80
Chevrolet             58
Land Rover            52
Jaguar                35
Jeep                  29
Datsun                28
MG                    27
Kia                   27
Volvo                 26
Fiat                  22
Porsche               17
Other Brands          15
Mitsubishi            14
MINI                   8
Force Motors           8
Mini                   5
Bajaj                  4
Lexus                  4
Bentley                4
Ambassador             3
Ssangyong              3
Isuzu                  3
Ashok Leyland          3
Lamborghini            2
Eicher Polaris         2
Opel                   2
Smart                  1
Rolls-Royce            1
Hindustan Motors       1
Mahindra Renault       1
Maserati               1
Premier                1
Daewoo                 1
Name: Brand, dtype: int64
```



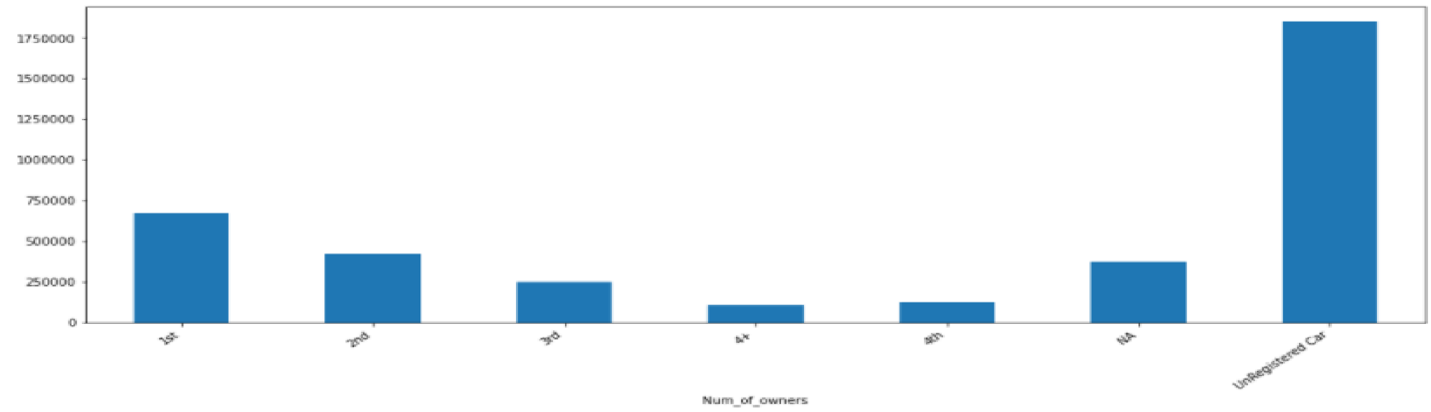
```
2017    423
2018    419
2016    392
2014    331
2015    329
       ...
1986      1
1988      1
1989      1
1995      1
1996      1
Name: Man_year, Length: 70, dtype: int64
```




```
Diesel            2473
Petrol            2295
CNG & Hybrids       82
NA                  57
LPG                 30
CNG                 14
CNG + Cng           14
Petrol + Cng         9
Automatic            4
Electric             4
Hybrid               3
Manual               2
Name: Fuel, dtype: int64
```




```
1st                 3619
2nd                 1000
3rd                  189
NA                   130
4th                   30
4+                    10
UnRegistered Car       9
Name: Num_of_owners, dtype: int64
```



1. Highest price is for Rolls-Royce and maximum count is for Maruti Suzuki and then Hyundai.

2. Highest price is for manufacturing year 2021, maximum count is for year 2017 and 2018.

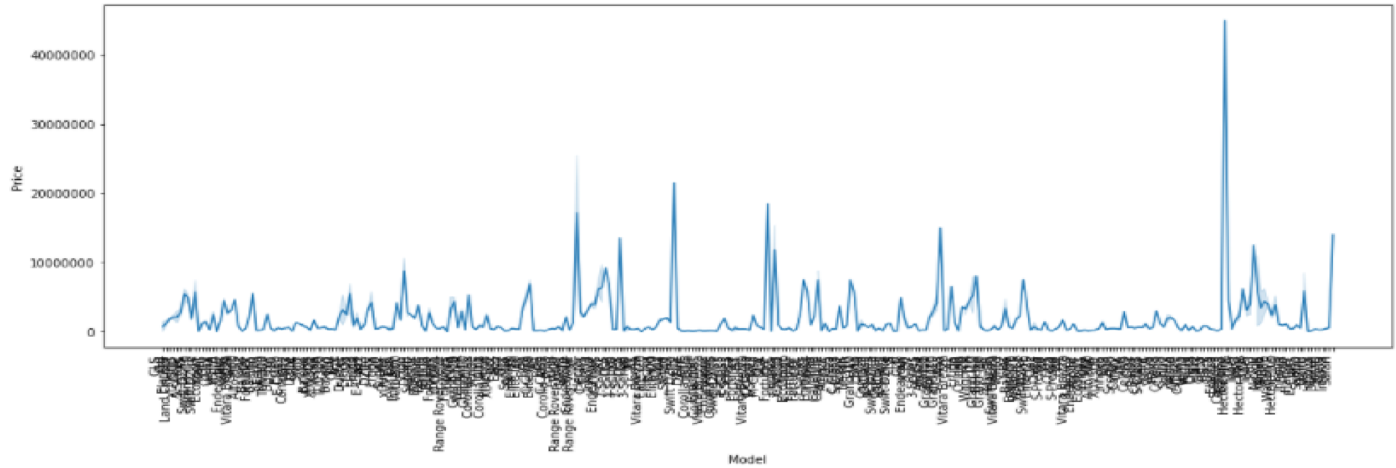
3. Highest price is for hybrid type of fuel; maximum count is for vehicle with fuel type Diesel and petrol.

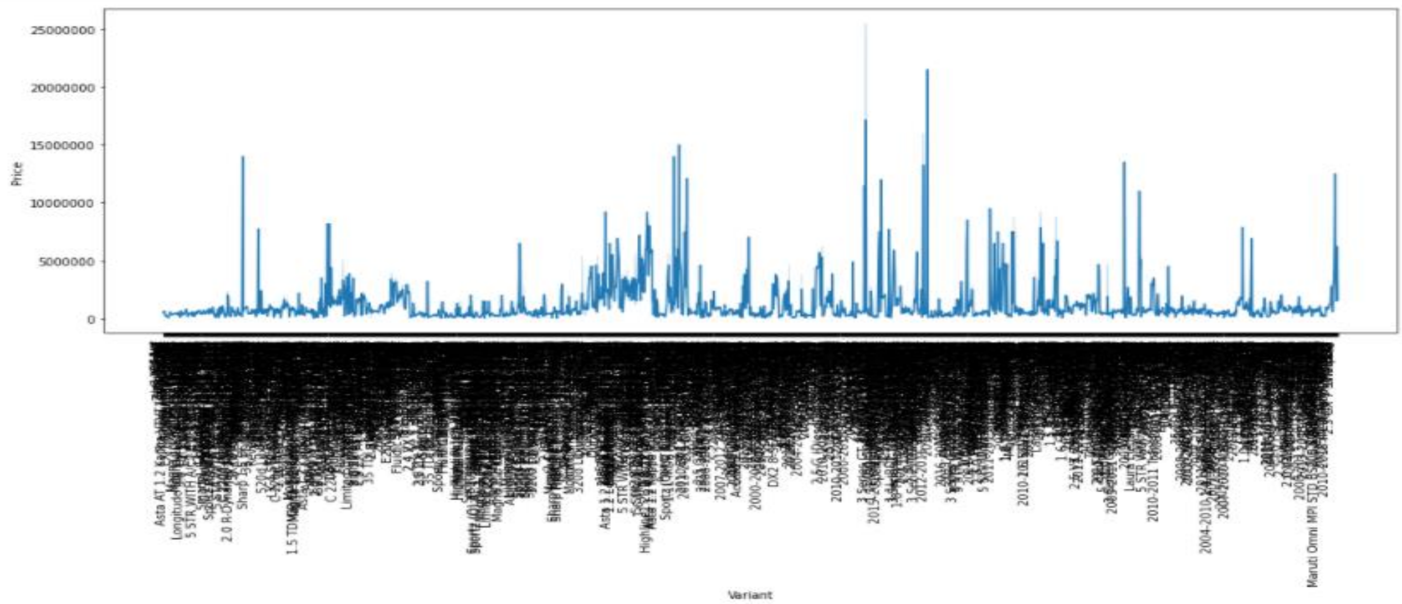
4. With respect to number of owners highest price is for Unregistered Car and maximum count is for 1st.

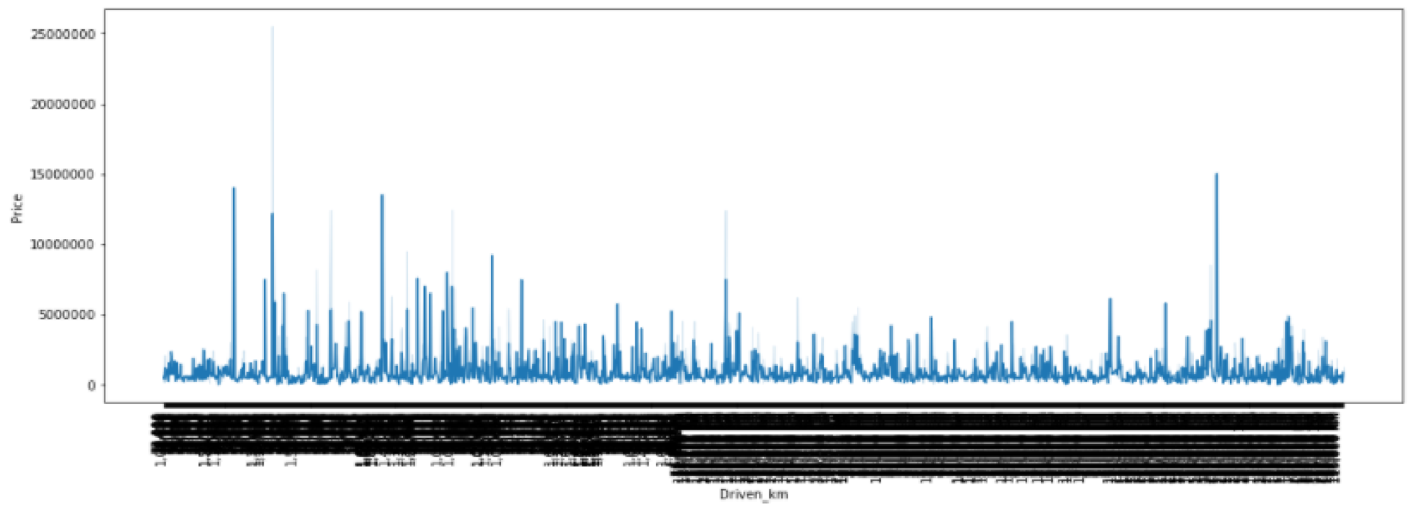
```
#for more than 70 keeping it in one list
dis_col=[]
for i in data:
    if data[i].nunique() > 70:
        dis_col.append(i)

print(dis_col)
```

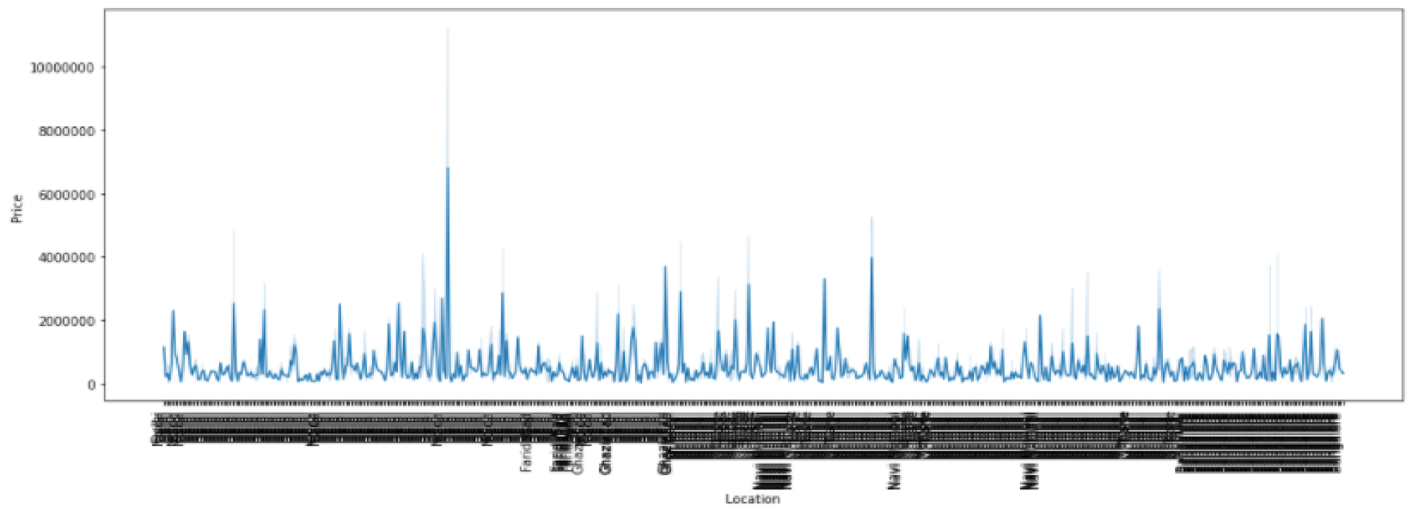
```
['Model', 'Variant', 'Driven_km', 'Location', 'Price']
```

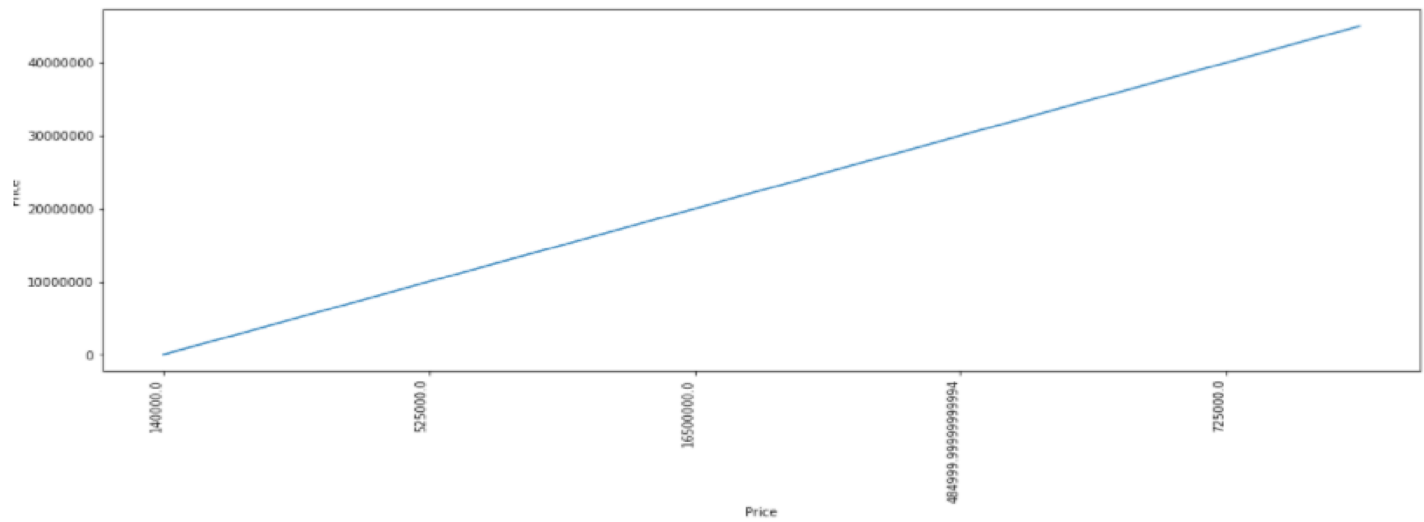
```
for i in dis_col:
    plt.figure(figsize=(18,6))
    a=sns.lineplot(x=data[i],y=data['Price'],palette='Tableau')
    #a.set_xticklabels(a.get_xticklabels(), rotation=90, ha="right",visible=True)
    a.set_xticklabels(labels=data[i],rotation=90, ha="right")
    #ax.ticklabel_format(useOffset=False, style='plain')
    plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
    plt.show()
```





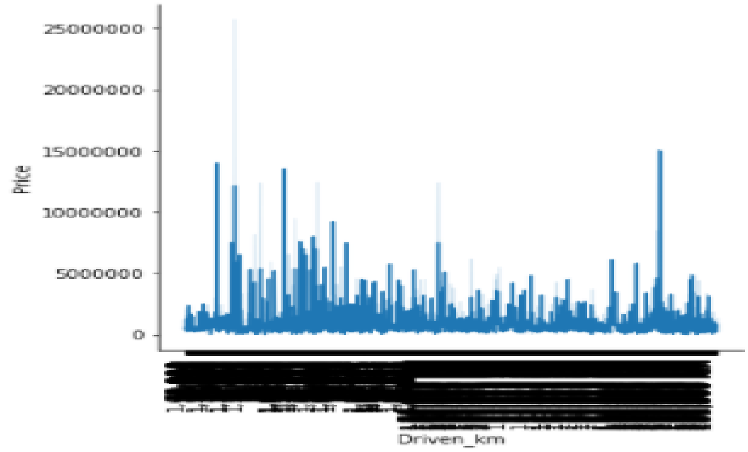

25000000
20000000
15000000
10000000
5000000
0
Price
Driven_km


10000000
8000000
6000000
4000000
2000000
0
Price
Location

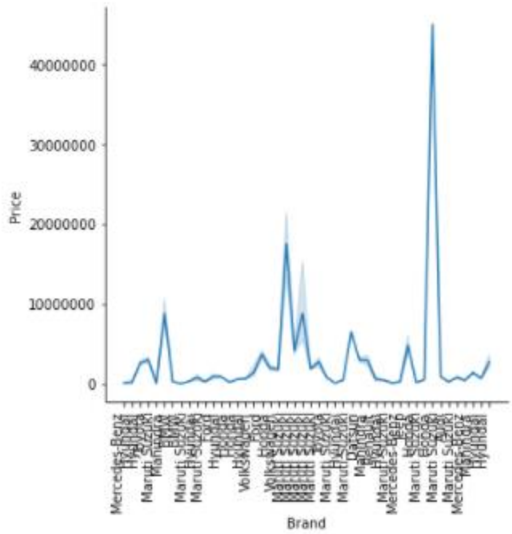

40000000
30000000
20000000
10000000
0
140000.0
525000.0
16500000.0
484999.9999999994
725000.0
Price

```
list=['Driven_km','Brand','Model','Variant','Fuel','Num_of_owners']
for i in list:
    plt.figure(figsize=(18,6))
    a=sns.relplot(x=i,y='Price',data=data,kind='line')
    a.set_xticklabels(labels=data[i],rotation=90, ha="right")
    plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
    plt.show()
```

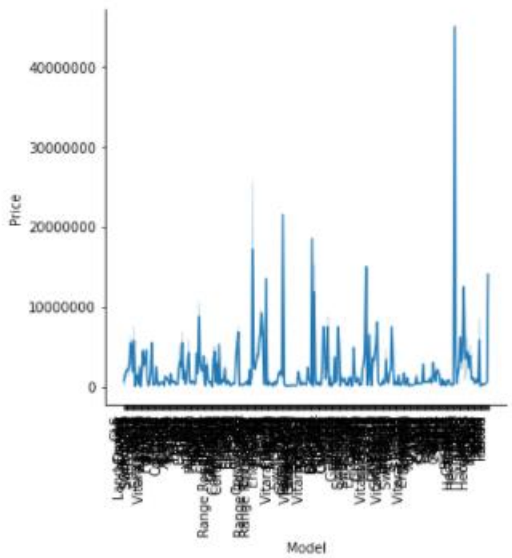
<Figure size 1296x432 with 0 Axes>

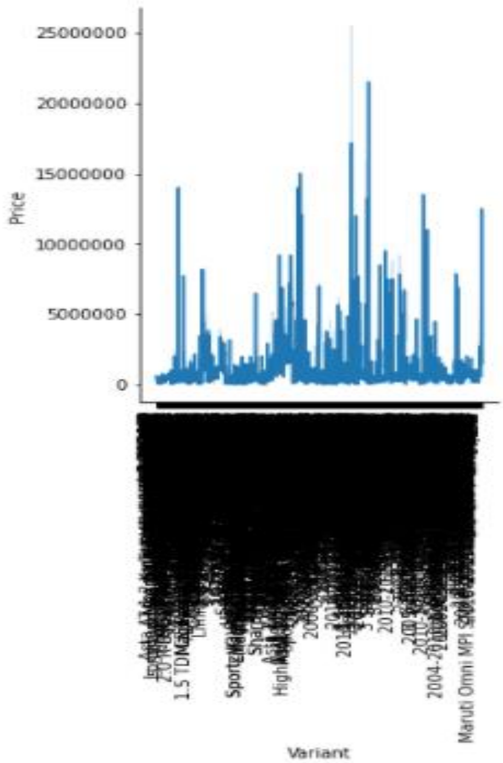


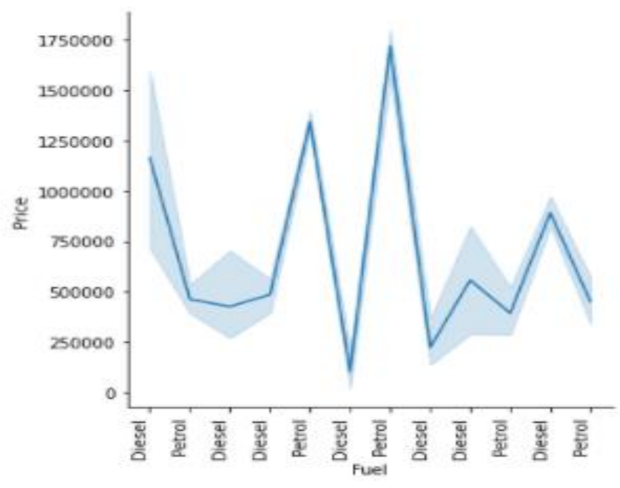
<Figure size 1296x432 with 0 Axes>

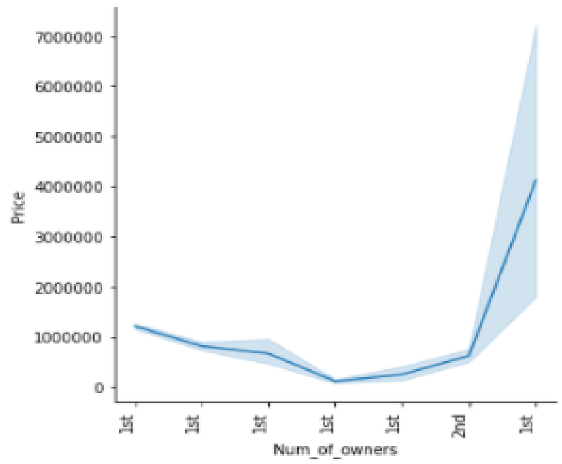


<Figure size 1296x432 with 0 Axes>









1. Maximum driven vehicles are for price less than 5000000.
2. Highest total of price Maruti Suzuki.
3. Maximum vehicles prices for models are with price less than 1000000.
4. Maximum vehicles prices for variants are with price less than 5000000.
5. Maximum vehicles prices for fuel are less with price than 100000.
6. Maximum vehicles for number of owners are with price less than 400000.

Let's Check the correlation now:

```
plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
plt.show()
```

Let's change the data type of other object type columns.

```
data.dtypes
```

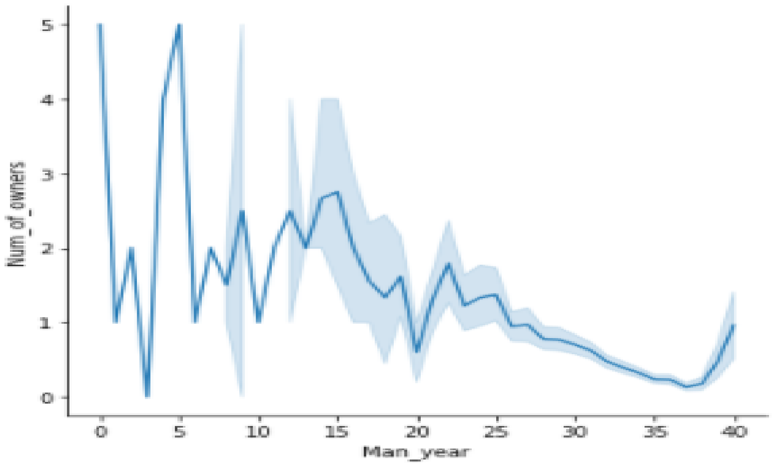
```
Brand              object
Model              object
Variant            object
Man_year           object
Driven_km          object
Fuel               object
Num_of_owners      object
Location           object
Price             float64
dtype: object
```

Encoding:

```python
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for i in data.columns:
    if data[i].dtypes=='object':
        data[i]=le.fit_transform(data[i].astype(str).values.reshape(-1,1))
```

```python
sns.relplot(x='Man_year', y='Num_of_owners',data=data, kind='line')
```

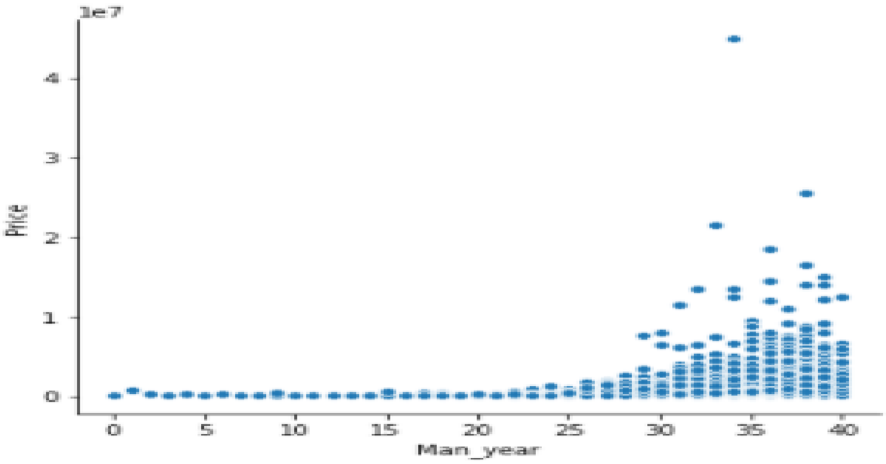
```
<seaborn.axisgrid.FacetGrid at 0x1764ffc92e0>
```



Num_of_owners.We can see negative relation between Man_year and Num_of_owners.

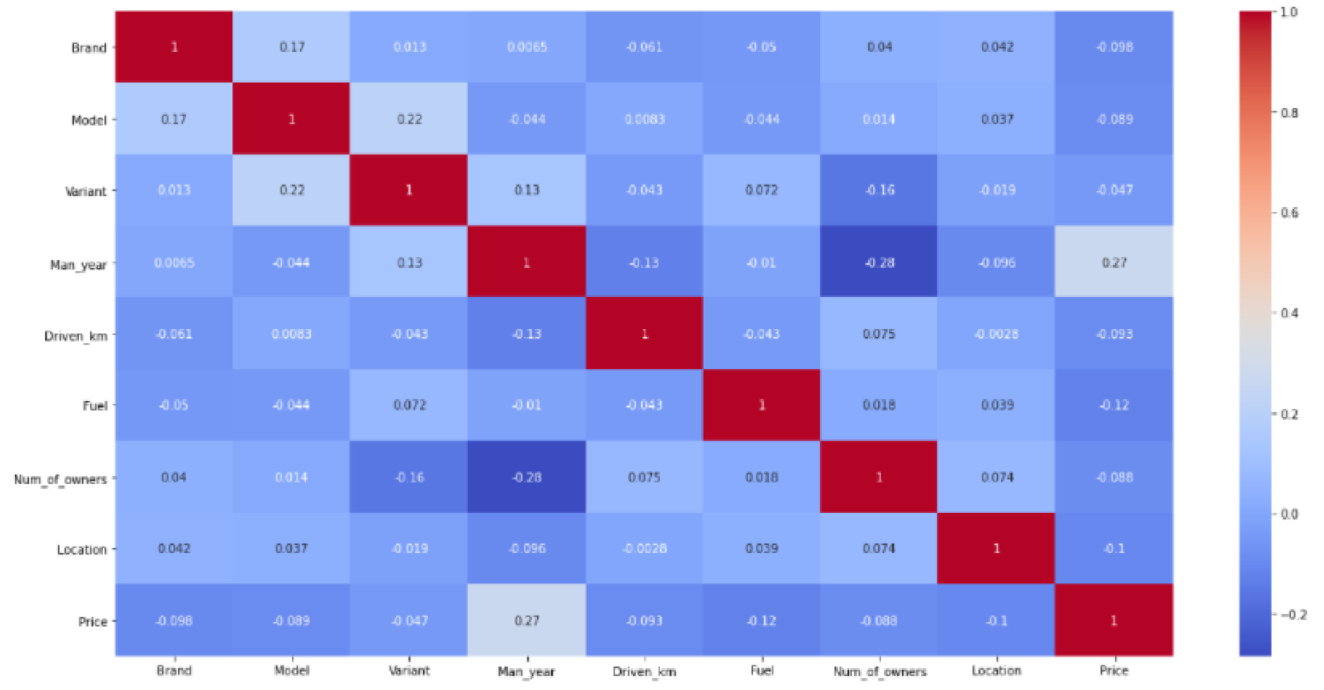
```python
sns.relplot(x='Man_year', y='Price',data=data, kind='scatter')
```

```
<seaborn.axisgrid.FacetGrid at 0x1764ffdff10>
```



Num_of_owners.We can see positive relation between Man_year and Price.

```
plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
plt.show()
```




```
cor_matrix=data.corr()
cor_matrix['Price'].sort_values(ascending=False)
```

```
Price            1.000000
Man_year         0.271550
Variant         -0.046876
Num_of_owners   -0.088368
Model           -0.089200
Driven_km       -0.092797
Brand           -0.097746
Location        -0.100956
Fuel            -0.119873
Name: Price, dtype: float64
```

Our target column Price is positively correlated with Man_year and has negative correlation with Fuel column.

```
x=data.drop('Price', axis=1)
y=data['Price']
print(x.shape)
print(y.shape)
```

```
(4987, 8)
(4987,)
```

Other than price all other columns object type hence Skewness and outliers removal is not required.

Model/s Development and Evaluation

Finding best random state:

```
maxAcc=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=i)
    Ln=LinearRegression()
    Ln.fit(x_train,y_train)
    pred=Ln.predict(x_test)
    acc=r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print("Best accuracy is ",maxAcc, " at Random State ",maxRS)
```

```
Best accuracy is  0.22141973573961382  at Random State  172
```

```
model=[LinearRegression(),AdaBoostRegressor(),ElasticNet(alpha=0.0001),
       KNeighborsRegressor(),DecisionTreeRegressor(), BaggingRegressor()]
for m in model:
    m.fit(x_train,y_train)
    #sc=m.score(x_train,y_train)
    predm=m.predict(x_test)
    acc=r2_score(y_test,predm)
    print('Accuracy Score of',m,'is:',acc * 100,"%")
    print('mean_absolute_error:',mean_absolute_error(y_test,predm))
    print('mean_squared_error:',mean_squared_error(y_test,predm))
    print('Root mean_squared_error:',np.sqrt(mean_squared_error(y_test,predm)))
    print("\n")
```

```
Accuracy Score of LinearRegression() is: 22.141973573961383 %
mean_absolute_error: 701875.4760323387
mean_squared_error: 1159971035503.4194
Root mean_squared_error: 1077019.5149129934


Accuracy Score of AdaBoostRegressor() is: -165.92033535763466 %
mean_absolute_error: 1799929.9956863571
mean_squared_error: 3961825144119.6504
Root mean_squared_error: 1990433.4061002017


Accuracy Score of ElasticNet(alpha=0.0001) is: 22.14196247471478 %
mean_absolute_error: 701875.4620424132
mean_squared_error: 1159971200866.0051
Root mean_squared_error: 1077019.5916816022


Accuracy Score of KNeighborsRegressor() is: 0.21274887216699012 %
mean_absolute_error: 674673.3298597195
mean_squared_error: 1486684499134.4526
Root mean_squared_error: 1219296.7231705547


Accuracy Score of DecisionTreeRegressor() is: -58.97573992489065 %
mean_absolute_error: 234181.44188376755
mean_squared_error: 2368506654041.3237
Root mean_squared_error: 1538995.3391876544


Accuracy Score of BaggingRegressor() is: 74.2901565804456 %
mean_absolute_error: 219260.5586888062
mean_squared_error: 383039168380.9426
Root mean_squared_error: 618901.5821444817
```

Highest accuracy for BaggingRegressor is 74.2901565804456 %

Hyper parameter tunning:

```
from sklearn.model_selection import RandomizedSearchCV
```

- LinearRegression

```
parameters1={'fit_intercept':[True, False],'normalize':[True, False],
             'copy_X':[True, False],'positive':[True, False],
             'n_jobs':range(0,20)}
```

```
RSV1=RandomizedSearchCV(LinearRegression(),parameters1,cv=5)
```

```
RSV1.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=LinearRegression(),
                   param_distributions={'copy_X': [True, False],
                                        'fit_intercept': [True, False],
                                        'n_jobs': range(0, 20),
                                        'normalize': [True, False],
                                        'positive': [True, False]})
```

```
RSV1.best_params_
```

```
{'positive': False,
 'normalize': False,
 'n_jobs': 18,
 'fit_intercept': True,
 'copy_X': False}
```

```
RSV1_pred=RSV1.best_estimator_.predict(x_test)
```

```
RSV1_pred
```

```
array([ 1747916.08983343,   263229.24292383,  1165360.39836314,
         809252.39214025,  1271622.05282929,  1658949.10077097,
         744666.45507292,  1388283.27281102,  1273926.7740467 ,
         959551.64674769,  1412782.55387153,   749922.11064204,
        1423160.25923321,   911554.49946795,   960014.53653004,
        1614168.07711112,   923865.18195259,  1003393.75492046,
        1184727.3187918 ,  1177712.29239486,  1744089.88821895,
        1133623.61556464,   276121.25944322,  1002468.22201975,
         553989.44839956,  1594374.17653606,   759245.08589626,
        1374504.41825787,  1524559.9905307 ,  1311035.57635628,
         386502.8244846 ,  1192683.81700453,  1199653.26150462,
        1683819.12246343,  1485715.80525799,   658712.18574233,
         743383.2713411 ,  1802333.27224981,  -229006.22914348,
        1632991.14512353,  1464795.5830619 ,   637559.68814063,
        1313214.9670704 ,  1120057.26201694,   780005.42403129,
        2022152.62745475,  1090074.03732448,  1436805.33893788,
        1003063.92856345,   422888.03156335,  1846272.47241273,
         859601.27473197,  1199992.90724048,   221361.94234664,
        1386523.40806566,  1279248.12833151,   926802.868534  ,
         703079.45129866,  1617060.37426289,   699298.17117957
```

```
score1 = RSV1.score(x_train,y_train)
```

- AdaBoostRegressor

```
param2={'n_estimators': [20, 50, 70, 100], 'learning_rate' : [0.01,0.05,0.1,0.3,1],
        'loss' : ['linear', 'square', 'exponential'],'random_state':range(0,20)}
```

```
RSV2=RandomizedSearchCV(AdaBoostRegressor(),param2,cv=5)
```

```
RSV2.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=AdaBoostRegressor(),
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1, 0.3,
                                                          1],
                                        'loss': ['linear', 'square',
                                                 'exponential'],
                                        'n_estimators': [20, 50, 70, 100],
                                        'random_state': range(0, 20)})
```

```
RSV2_pred=RSV2.best_estimator_.predict(x_test)
```

```
RSV2.best_params_
```

```
{'random_state': 12,
 'n_estimators': 50,
 'loss': 'exponential',
 'learning_rate': 0.01}
```

```
RSV2_pred
```

```
array([ 571533.15454545,  434265.0854527 ,  571533.15454545,
        571533.15454545, 1121702.73149171, 2286663.73302469,
       1155860.11001236,  940466.33831217,  922661.54643629,
        922661.54643629, 1873552.19689119,  571533.15454545,
       1121702.73149171,  455227.26407613, 2286663.73302469,
        856109.58232676,  455227.26407613,  922661.54643629,
       2286663.73302469, 1116948.75767544, 1121702.73149171,
        571533.15454545,  483422.7077597 ,  483422.7077597 ,
        437198.58       , 1898824.50961538,  483422.7077597 ,
        856109.58232676, 2286663.73302469, 1121702.73149171,
        434265.0854527 , 2286663.73302469,  856109.58232676,
       1121702.73149171, 2286663.73302469,  922661.54643629,
        434265.0854527 , 4130053.18085106,  434265.0854527 ,
       1873552.19689119, 1121702.73149171,  434265.0854527 ,
        455227.26407613, 1094237.71238938, 2286663.73302469,
       1116948.75767544, 1121702.73149171,  922661.54643629,
       1116948.75767544,  437198.58       , 2301426.90363349,
        455227.26407613,  856109.58232676,  434265.0854527 ,
        571533.15454545, 1116948.75767544, 2286663.73302469,
        856109.58232676, 2286663.73302469,  922661.54643629
```

```
score2 = RSV2.score(x_train,y_train)
```

- ElasticNet

```
parameters3={'fit_intercept':[True, False],'normalize':[True, False],'precompute':[True, False],
            'copy_X':[True, False],'positive':[True, False],'warm_start':[True, False],
            'random_state':range(0,20),'l1_ratio':[0.5,1], 'selection':['cyclic','random']}
```

```
RSV3=RandomizedSearchCV(ElasticNet(alpha=0.0001),parameters3,cv=5)
```

```
RSV3.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=ElasticNet(alpha=0.0001),
                   param_distributions={'copy_X': [True, False],
                                        'fit_intercept': [True, False],
                                        'l1_ratio': [0.5, 1],
                                        'normalize': [True, False],
                                        'positive': [True, False],
                                        'precompute': [True, False],
                                        'random_state': range(0, 20),
                                        'selection': ['cyclic', 'random'],
                                        'warm_start': [True, False]})
```

```
RSV3.best_params_
```

```
{'warm_start': True,
 'selection': 'cyclic',
 'random_state': 6,
 'precompute': True,
 'positive': False,
 'normalize': False,
 'l1_ratio': 1,
 'fit_intercept': True,
 'copy_X': True}
```

```
RSV3_pred=RSV3.best_estimator_.predict(x_test)
```

```
RSV3_pred
```

```
array([ 1747916.0897399 ,   263229.24297752,  1165360.39831502,
         809252.3920807 ,  1271622.05275221,  1658949.10077881,
         744666.45500513,  1388283.27284651,  1273926.77405025,
         959551.64673341,  1412782.55378226,   749922.11057299,
        1423160.25923372,   911554.49946397,   960014.53654448,
        1614168.0770492 ,   923865.1819102 ,  1003393.75485353,
        1184727.31879237,  1177712.29237641,  1744089.88815166,
        1133623.61547347,   276121.2594154 ,  1002468.22195484,
         553989.4483403 ,  1594374.17644071,   759245.08587276,
        1374504.41818527,  1524559.99050475,  1311035.57628947,
         386502.82455245,  1192683.81694144,  1199653.26146225,
        1683819.12239893,  1485715.80521203,   658712.18573737,
         743383.27135418,  1802333.27215662,  -229006.22904679,
        1632991.14502327,  1464795.582986  ,   637559.68806157,
        1313214.96707952,  1120057.26195812,   780005.42402504,
        2022152.62736117,  1090074.03731233,  1436805.33893312,
        1003063.9285412 ,   422888.03149869,  1846272.47233453,
         859601.27479235,  1199992.90717413,   221361.94232464,
        1386523.40795419,  1279248.12827526,   926802.86853201,
         703079.45177369,  1617060.37420422,   699298.17117312
```

```
score3 = RSV3.score(x_train,y_train)
```

- KNeighborsRegressor

```
parameters4={'weights':['uniform','distance'],'algorithm':['auto','ball_tree','kd_tree','brute'],
            'n_jobs':range(0,20)}
```

```
RSV4=RandomizedSearchCV(KNeighborsRegressor(),parameters4,cv=5)
```

```
RSV4.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=KNeighborsRegressor(),
                   param_distributions={'algorithm': ['auto', 'ball_tree',
                                                      'kd_tree', 'brute'],
                                        'n_jobs': range(0, 20),
                                        'weights': ['uniform', 'distance']})
```

```
RSV4.best_params_
```

```
{'weights': 'distance', 'n_jobs': 15, 'algorithm': 'ball_tree'}
```

```
RSV4_pred=RSV4.best_estimator_.predict(x_test)
```

```
RSV4_pred
```

```
        889519.21572261, 1373520.42450881,  483166.21398329,
        279507.74922707,  404384.23116967, 1584215.04133504,
        439280.78771271, 1095260.83593294, 2250474.47870666,
        825588.50658281, 1114212.76043067, 1001683.96560702,
        519567.99471342,  475000.        ,  906368.05515435,
        480000.        , 1570344.63456708,  852860.43519441,
       1771203.20105286, 1213862.69274526,  848141.5646022 ,
       1665487.33661886,  635275.91413163, 5768347.9230576 ,
        902831.28608295, 1313689.077691  ,  496526.16674175,
       1348099.9654708 ,  572469.48562806,  591428.72534861,
       3639314.1435085 ,  356759.93425032,  743983.74138301,
        424919.85464224,  446579.36328044,  661754.94103798,
        831464.99057611,  434845.73700112, 1641925.48455017,
        671715.94470769,  386910.83569149,  667927.62205928,
        271785.18852304, 1740000.        ,  127280.45709579,
       2106270.2053562 ,  753623.61881982,  676497.61642286,
       1337674.97154138,  565423.65551979,  827833.77788572,
        318175.63605683, 7213302.57156838, 1165507.75093601,
        859496.59203827, 1386597.73849173,  775000.        ,
        426909.05195072,  144910.85546114, 1122867.60842781,
```

```
score4 = RSV4.score(x_train,y_train)
```

- DecisionTreeRegressor

```
parameters5={'criterion':['mse', "friedman_mse", "mae", 'poisson'],'splitter':['best','random'],
            'max_features':['auto','sqrt','log2'],'random_state':range(0,20)}
```

```
RSV5=RandomizedSearchCV(DecisionTreeRegressor(),parameters5,cv=5)
```

```
RSV5.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=DecisionTreeRegressor(),
                   param_distributions={'criterion': ['mse', 'friedman_mse',
                                                      'mae', 'poisson'],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'random_state': range(0, 20),
                                        'splitter': ['best', 'random']})
```

```
RSV5.best_params_
```

```
{'splitter': 'best',
 'random_state': 17,
 'max_features': 'auto',
 'criterion': 'mse'}
```

```
RSV5_pred=RSV5.best_estimator_.predict(x_test)
```

```
RSV5_pred
```

```
array([6.250000e+05, 2.650000e+05, 6.500000e+05, 5.800000e+05,
       1.299000e+06, 1.500000e+04, 1.100000e+05, 1.500000e+06,
       4.850000e+05, 4.650000e+05, 9.250000e+05, 3.250000e+05,
       5.250000e+05, 1.950000e+05, 5.450000e+05, 2.750000e+05,
       3.410000e+05, 1.079000e+06, 8.100000e+05, 6.910000e+05,
       1.149000e+06, 5.000000e+05, 3.300000e+05, 4.420000e+05,
       2.580000e+05, 1.450000e+06, 2.750000e+05, 2.900000e+05,
       4.320000e+05, 1.100000e+06, 3.590000e+05, 1.950000e+06,
       9.950000e+05, 8.250000e+05, 9.750000e+05, 3.950000e+05,
       3.500000e+05, 2.485000e+06, 1.800000e+05, 1.100000e+06,
       9.250000e+05, 1.250000e+05, 1.800000e+05, 5.750000e+05,
       7.600000e+05, 5.950000e+05, 1.125000e+06, 3.200000e+05,
       8.750000e+05, 1.980000e+05, 2.950000e+06, 4.650000e+05,
       9.450000e+05, 2.750000e+05, 4.750000e+05, 6.700000e+05,
       9.750000e+05, 5.000000e+05, 5.900000e+05, 8.450000e+05,
       1.180000e+06, 3.500000e+05, 9.750000e+05, 4.450000e+06,
       5.500000e+04, 6.400000e+05, 5.500000e+06, 1.750000e+06,
       7.500000e+05, 9.900000e+04, 1.125000e+06, 3.200000e+06,
       1.750000e+06, 2.690000e+06, 4.800000e+05, 2.375000e+06,
       3.350000e+05, 9.250000e+05, 3.850000e+05, 8.750000e+05
```

```
score5 = RSV5.score(x_train,y_train)
```

- BaggingRegressor

```
#creating  parameter list to pass in RandomizedSearchCV
param6={'base_estimator':[None],'n_estimators':[0,2,5,10],
      'bootstrap':[True,False],'oob_score':[True,False],'warm_start':[True,False],
       'bootstrap_features':[True,False],'random_state':range(0,20)}
```

```
RSV6=RandomizedSearchCV(BaggingRegressor(),param6,cv=5)
```

```
RSV6.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=BaggingRegressor(),
                   param_distributions={'base_estimator': [None],
                                        'bootstrap': [True, False],
                                        'bootstrap_features': [True, False],
                                        'n_estimators': [0, 2, 5, 10],
                                        'oob_score': [True, False],
                                        'random_state': range(0, 20),
                                        'warm_start': [True, False]})
```

```
RSV6.best_params_
```

```
{'warm_start': True,
 'random_state': 2,
 'oob_score': False,
 'n_estimators': 10,
 'bootstrap_features': True,
 'bootstrap': False,
 'base_estimator': None}
```

```
RSV6_pred=RSV6.best_estimator_.predict(x_test)
```

```
RSV6_pred
```

```
array([ 498000.        ,  198800.        ,  686923.07692308,
        703923.07692308, 1099400.        ,   76666.66666667,
        419611.        , 1501250.        ,  493000.        ,
        467500.        ,  932500.        ,  456432.25483871,
        525000.        ,  480100.        ,  546800.        ,
        291333.33333333,  348833.33333333, 1151200.        ,
        810000.        ,  666900.        , 1152800.        ,
        678000.        ,  318349.9       ,  443875.        ,
        258000.        , 2238000.        ,  468499.9       ,
        288500.        ,  351450.        , 1291000.        ,
        364800.        , 1977500.        ,  814900.        ,
        724500.        , 1065500.        ,  395000.        ,
        538500.        , 2233833.33333333,   95400.        ,
       1117833.33333333,  904228.57142857, 1070675.        ,
        246400.        ,  573666.66666667,  665678.26086957,
        719500.        , 1096025.        ,  320000.        ,
        707499.8       ,  363511.2375    , 2950000.        ,
        604840.        ,  945000.        ,  275000.        ,
        437000.        ,  801785.71428571,  959666.66666667
```

```
score6 = RSV6.score(x_train,y_train)
```

After performing RandomizedSearchCV method accuracy

```
print("Accuracy for LinearRegression is ",score1*100,"%\n")
print("Accuracy for AdaBoostRegressor is ",score2*100,"%\n")
print("Accuracy for ElasticNet(alpha=0.0001) is ",score3*100,"%\n")
print("Accuracy for KNeighborsRegressor is ",score4*100,"%\n")
print("Accuracy for DecisionTreeRegressor is ",score5*100,"%\n")
print("Accuracy for BaggingRegressor is ",score6*100,"%")
```

```
Accuracy for LinearRegression is  10.205992452519341 %

Accuracy for AdaBoostRegressor is  25.97032009172696 %

Accuracy for ElasticNet(alpha=0.0001) is  10.205992452519352 %

Accuracy for KNeighborsRegressor is  99.99998971969106 %

Accuracy for DecisionTreeRegressor is  99.99998971969106 %

Accuracy for BaggingRegressor is  99.95804638877678 %
```

:

Selecting BaggingRegressor as final model for saving, as it was having good accuracy with model as well.

Saving best model:

```
import joblib
joblib.dump(RSV6,"RSCPR.obj")
```

```
['RSCPR.obj']
```

```
RSVfile=joblib.load("RSCPR.obj")
RSVfile.predict(x_test)
```

```
array([ 498000.        ,  198800.        ,  686923.07692308,
        703923.07692308, 1099400.        ,   76666.66666667,
        419611.        , 1501250.        ,  493000.        ,
        467500.        ,  932500.        ,  456432.25483871,
        525000.        ,  480100.        ,  546800.        ,
        291333.33333333,  348833.33333333, 1151200.        ,
        810000.        ,  666900.        , 1152800.        ,
        678000.        ,  318349.9       ,  443875.        ,
        258000.        , 2238000.        ,  468499.9       ,
        288500.        ,  351450.        , 1291000.        ,
        364800.        , 1977500.        ,  814900.        ,
        724500.        , 1065500.        ,  395000.        ,
        538500.        , 2233833.33333333,   95400.        ,
       1117833.33333333,  904228.57142857, 1070675.        ,
        246400.        ,  573666.66666667,  665678.26086957,
        719500.        , 1096025.        ,  320000.        ,
        707499.8       ,  363511.2375    , 2950000.        ,
        604840.        ,  945000.        ,  275000.        ,
        437000.        ,  801785.71428571,  959666.66666667,
        425499.7       ,  929300.        ,  686525.38205128
```

CONCLUSION

In the Car Price Prediction, I have extracted Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location and at last target variable Price of the car from different websites then saved the extracted csv file into one excel sheet. Our dataset mainly consists of 10 columns and 5073 rows. We have one column named Unnamed: 0 however that is not required for further analysis.

Then I did some preprocessing like dropping replacing '-' with NA , Converted price column into numeric, removed alphabetes from Driven_km and Man_year and then removed duplication of values. Then I have performed some visualization. After encoding object type variables, we have check for correlation.

Trying finding out best random random state and then used same for model building. Used 5 methods for model building then with the help of RandomizedSearchCV I have tried to improve accuracy. Finally, I decided to go ahead with the BaggingRegressor and saved the model