FLIP ROBO

# NAME OF THE PROJECT

CAR PRICE PREDICTION

## SUBMITTED BY

**RAUSHAN KUMAR**

# ABSTRACT

The price of a new car in the industry is fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthy. But, due to the increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. Existing System includes a process where a seller decides a price randomly and buyer has no idea about the car and it’s value in the present day scenario. In fact, seller also has no idea about the car’s existing value or the price he should be selling the car at. To overcome this problem we have developed a model which will be highly effective. Regression Algorithms are used because they provide us with continuous value as an output and not a categorized value. Because of which it will be possible to predict the actual price a car rather than the price range of a car. User Interface has also been developed which acquires input from any user and displays the Price of a car according to user’s inputs.

# INTRODUCTION

Determining whether the listed price of a used car is a challenging task, due to the many factors that drive a used vehicle’s price on the market. The focus of this project is developing machine learning models that can accurately predict the price of a used car based on its features, in order to make informed purchases.

In this project, various learning methods on a dataset consisting of the sale prices of different makes and models are implemented and evaluated. We will compare the performance of various machine learning algorithms like Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor Random Forest Regressor and choose the best out of it. Depending on various parameters we will determine the price of the car. Regression Algorithms are used because they provide us with continuous value as an output and not a categorized value because of which it will be possible to predict the actual price a car rather than the price range of a car. User Interface has also been developed which acquires input from any user and displays the Price of a car according to user’s inputs.

# Analytical Problem Framing

In this project,we used the different mathematical and statical functions to describe the data more efficiently.

1. Isnull():This function is used to identify whether the data set have any null values or not.

2. describe():This function give all stastical summary of data set.

For exp:count,mean,median,max,min values

3. Shape():This function tells us how many rows and columns present in the dataset.

Hardware and Software Requirements and Tools Used

`We used jupyter notebook for this project.

Following libraries are used:

1) Pandas:used for mathematical and statical analysis of data. For example:

- pandas.read_csv():used to read csv file
- pandas.Dataframe():passed the data to dataframe so we can perform different operations on data

2) Seaborn:used for visualization

- Heatmap: used to visualize colinearity between variables
- Distplot: used to visualize distribution of dataset
- Countplot: used to visualize categorical data

# Data Sources:

Data for used car price prediction project is collected from different car selling websites such as olx,cars24, cardekho etc. This data set contains columns such as driven kilometers, number of owners, model name, company name, location, price etc.

Following figure shows some values of data set:

| | Unnamed: 0 | Fuel | Driven_kilometers | Num_of_owners | Transmission | Location | Name | Year | Company | Price1 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Petrol | 23 km | 1st Owner | NaN | DL-4C | Maruti OMNI E | 2014 | Maruti | 2,00,199 |
| 1 | 1 | Petrol | 12,535 km | 1st Owner | MANUAL | DL-12 | Maruti Alto 800 | 2014 | Maruti | 3,21,599 |
| 2 | 2 | Petrol | 2,589 km | 1st Owner | NaN | UP-32 | Hyundai VENUE S | 2021 | Hyundai | 8,08,699 |
| 3 | 3 | Petrol | 40,184 km | 1st Owner | MANUAL | DL-8C | Maruti Alto K10 | 2013 | Maruti | 2,42,299 |
| 4 | 4 | Petrol | 9,217 km | 1st Owner | MANUAL | DL-8C | Maruti Alto 800 | 2015 | Maruti | 2,76,199 |
| 5 | 5 | Petrol | 31,999 km | 1st Owner | MANUAL | DL-1C | Maruti Swift LXI | 2012 | Maruti | 3,22,399 |
| 6 | 6 | Petrol | 19,415 km | NaN | NaN | DL-13 | Honda Brio 1.2 | 2012 | Honda | 2,83,799 |
| 7 | 7 | Petrol | 22,836 km | NaN | NaN | UP-16 | Hyundai Grand i10 | 2014 | Hyundai | 4,11,999 |
| 8 | 8 | Petrol | 11,691 km | NaN | NaN | DL-12 | Maruti Alto K10 | 2017 | Maruti | 3,81,599 |
| 9 | 9 | Petrol | 24,353 km | 1st Owner | MANUAL | DL-4C | Maruti Alto K10 | 2011 | Maruti | 2,34,999 |
| 10 | 10 | Petrol | 12,749 km | NaN | NaN | HR-51 | Hyundai Eon MAGNA | 2015 | Hyundai | 3,22,599 |
| 11 | 11 | Petrol | 39,300 km | NaN | NaN | DL-4C | Maruti Zen Estilo | 2011 | Maruti | 1,89,399 |
| 12 | 12 | Petrol | 34,364 km | NaN | NaN | DL-4C | Volkswagen Polo HIGHLINE1.2L | 2012 | Volkswagen | 4,02,499 |
| 13 | 13 | Petrol | 20,039 km | NaN | NaN | DL-1C | Tata Nano TWIST | 2016 | Tata | 2,10,599 |
| 14 | 14 | Petrol | 7,610 km | NaN | NaN | HR-26 | Maruti Alto LXI | 2020 | Maruti | 3,75,099 |
| 15 | 15 | Petrol | 13,899 km | 1st Owner | NaN | DL-2C | Hyundai Grand i10 | 2018 | Hyundai | 4,59,899 |
| 16 | 16 | Petrol | 33,175 km | 1st Owner | NaN | DL-7C | Maruti Alto K10 | 2012 | Maruti | 2,35,799 |
| 17 | 17 | Petrol | 5,645 km | 1st Owner | NaN | HR-98 | Maruti Baleno SIGMA | 2020 | Maruti | 5,80,999 |
| 18 | 18 | Petrol | 12,139 km | 1st Owner | NaN | DL-2C | Maruti Alto K10 | 2017 | Maruti | 3,53,599 |
| 19 | 19 | Petrol | 10,608 km | 1st Owner | NaN | DL-12 | Maruti Alto 800 | 2017 | Maruti | 3,19,199 |

1)Fuel: This column provides information about what type offuel is used for car. E.g. petrol, diesel, cng etc

2)Driven kilometers: This column gives information about how many kilometers car has been driven.

3) Num_of_owners:This columns tells about how many people used car.

4) Transmission: The transmission is a basic part of your car. It is mounted directly on the engine and converts the engine's combustion power to momentum which drives the wheels

It has two types:

- Manual: Vehicles with a manual or standard transmission are typically called **stick shifts**. The driver uses a stick shift to manually change the gears as they accelerate and decelerate their vehicle
- Automatic: According to State Farm, an automatic car is an automobile with an automatic transmission that doesn't require a driver to shift gears manually. Transmissions, also known as gearboxes, help to direct the rotational force and speed of a car. Therefore, automatic transmissions switch gear ratios as the vehicle moves

5) Location: This column gives information about location atwhere car is available.

6) Name: This column provides information about model name ofthe car.

7) Name: This column gives information about  company name ofthe car.

8) Year: This column tells that how many years car is old.

9)Price: This column provides the price of the car.Following fig shows data type of each column:

```
: 1 df.dtypes

: Unnamed: 0             int64
  Fuel                  object
  Driven_kilometers     object
  Num_of_owners         object
  Transmission          object
  Location              object
  Name                  object
  Year                  object
  Company               object
  Price1                object
  dtype: object
```

Above fig shows the data types of each column. All columns have object type values, but there is need to change data type ofsome columns such as price, years, num_of_owners etc.

# Data Analysis:

Data analysis involves manipulating, transforming, and visualizing data in order to infer meaningful insights from the results. Individuals, businesses, and even governments often take direction based on these insights. In Data analysis, we have check data types, missing values, and many more things. so let's do it one by one.

First importing all necessary libraries one by one.

```
.]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso,Ridge
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
```

## Load the data set:

```
ds=pd.read_csv("Used_car_price1.csv")
```

```
ds.head(20)
```

| | Unnamed: 0 | Fuel | Driven_kilometers | Num_of_owners | Transmission | Location | Name | Year | Company | Price1 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Petrol | 23 km | 1st Owner | NaN | DL-4C | Maruti OMNI E | 2014 | Maruti | 2,00,199 |
| 1 | 1 | Petrol | 12,535 km | 1st Owner | MANUAL | DL-12 | Maruti Alto 800 | 2014 | Maruti | 3,21,599 |
| 2 | 2 | Petrol | 2,589 km | 1st Owner | NaN | UP-32 | Hyundai VENUE S | 2021 | Hyundai | 8,08,699 |
| 3 | 3 | Petrol | 40,184 km | 1st Owner | MANUAL | DL-8C | Maruti Alto K10 | 2013 | Maruti | 2,42,299 |
| 4 | 4 | Petrol | 9,217 km | 1st Owner | MANUAL | DL-8C | Maruti Alto 800 | 2015 | Maruti | 2,76,199 |
| 5 | 5 | Petrol | 31,999 km | 1st Owner | MANUAL | DL-1C | Maruti Swift LXI | 2012 | Maruti | 3,22,399 |
| 6 | 6 | Petrol | 19,415 km | NaN | NaN | DL-13 | Honda Brio 1.2 | 2012 | Honda | 2,83,799 |
| 7 | 7 | Petrol | 22,836 km | NaN | NaN | UP-16 | Hyundai Grand i10 | 2014 | Hyundai | 4,11,999 |
| 8 | 8 | Petrol | 11,691 km | NaN | NaN | DL-12 | Maruti Alto K10 | 2017 | Maruti | 3,81,599 |
| 9 | 9 | Petrol | 24,353 km | 1st Owner | MANUAL | DL-4C | Maruti Alto K10 | 2011 | Maruti | 2,34,999 |
| 10 | 10 | Petrol | 12,749 km | NaN | NaN | HR-51 | Hyundai Eon MAGNA | 2015 | Hyundai | 3,22,599 |
| 11 | 11 | Petrol | 39,300 km | NaN | NaN | DL-4C | Maruti Zen Estilo | 2011 | Maruti | 1,89,399 |
| 12 | 12 | Petrol | 34,364 km | NaN | NaN | DL-4C | Volkswagen Polo HIGHLINE1.2L | 2012 | Volkswagen | 4,02,499 |
| 13 | 13 | Petrol | 20,039 km | NaN | NaN | DL-1C | Tata Nano TWIST | 2016 | Tata | 2,10,599 |
| 14 | 14 | Petrol | 7,610 km | NaN | NaN | HR-26 | Maruti Alto LXI | 2020 | Maruti | 3,75,099 |
| 15 | 15 | Petrol | 13,899 km | 1st Owner | NaN | DL-2C | Hyundai Grand i10 | 2018 | Hyundai | 4,59,899 |
| 16 | 16 | Petrol | 33,175 km | 1st Owner | NaN | DL-7C | Maruti Alto K10 | 2012 | Maruti | 2,35,799 |
| 17 | 17 | Petrol | 5,645 km | 1st Owner | NaN | HR-98 | Maruti Baleno SIGMA | 2020 | Maruti | 5,80,999 |
| 18 | 18 | Petrol | 12,139 km | 1st Owner | NaN | DL-2C | Maruti Alto K10 | 2017 | Maruti | 3,53,599 |
| 19 | 19 | Petrol | 10,608 km | 1st Owner | NaN | DL-12 | Maruti Alto 800 | 2017 | Maruti | 3,19,199 |

## Checking shape of the data set:

```
ds.shape
```

```
(6712, 10)
```

Data set have 6712 rows and 10 columns.

## Info about data set:

```
1 ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6712 entries, 0 to 6711
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Unnamed: 0         6712 non-null   int64 
 1   Fuel               6607 non-null   object
 2   Driven_kilometers  6695 non-null   object
 3   Num_of_owners      6101 non-null   object
 4   Transmission       6129 non-null   object
 5   Location           6684 non-null   object
 6   Name               6712 non-null   object
 7   Year               6712 non-null   object
 8   Company            6712 non-null   object
 9   Price1             6692 non-null   object
dtypes: int64(1), object(9)
memory usage: 524.5+ KB
```

Data set have 9 columns and 6712 features of object type.

Some columns have missing values.

## Checking data types of data set:

```
1 df.dtypes
```

```
Unnamed: 0            int64
Fuel                 object
Driven_kilometers    object
Num_of_owners        object
Transmission         object
Location             object
Name                 object
Year                 object
Company              object
Price1               object
dtype: object
```

All columns are object type. There is need to change the data type of some columns such as price.

## Checking missing values:

```
1  ds.isnull().sum()
```

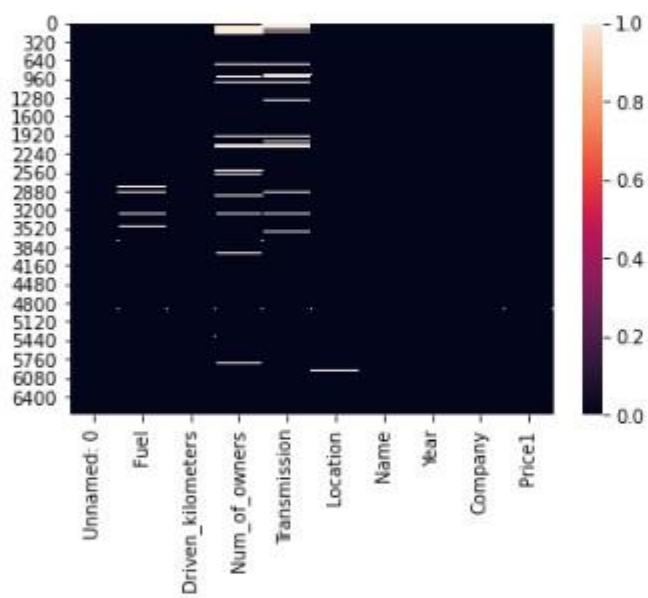
```
Unnamed: 0            0
Fuel                105
Driven_kilometers    17
Num_of_owners       611
Transmission        583
Location             28
Name                  0
Year                  0
Company               0
Price1               20
dtype: int64
```

Above fig shows that many columns have missing values.

And price column,which is target variable also has missing values, so we have remove that rows in which price is missing.

```
1  sns.heatmap(ds.isnull())
```

```
<AxesSubplot:>
```



Above heatmap also shows that there are missing values in data set.

There is need to delete those rows in which price values are missing.

```
df=pd.DataFrame(ds)
```

```
df=df[df['Price1'].notna()]
df.head()
```

| | Unnamed: 0 | Fuel | Driven_kilometers | Num_of_owners | Transmission | Location | Name | Year | Company | Price1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | Petrol | 23 km | 1st Owner | NaN | DL-4C | Maruti OMNI E | 2014 | Maruti | 2,00,199 |
| **1** | 1 | Petrol | 12,535 km | 1st Owner | MANUAL | DL-12 | Maruti Alto 800 | 2014 | Maruti | 3,21,599 |
| **2** | 2 | Petrol | 2,589 km | 1st Owner | NaN | UP-32 | Hyundai VENUE S | 2021 | Hyundai | 8,08,699 |
| **3** | 3 | Petrol | 40,184 km | 1st Owner | MANUAL | DL-8C | Maruti Alto K10 | 2013 | Maruti | 2,42,299 |
| **4** | 4 | Petrol | 9,217 km | 1st Owner | MANUAL | DL-8C | Maruti Alto 800 | 2015 | Maruti | 2,76,199 |

```
df.shape
```

Now data set have no missing values in price column.

Now let’s look into each column separately.

```
1 df['Company'].value_counts()
```

```
Maruti           2540
Hyundai          1200
Toyota            475
Honda             430
Mahindra          401
Ford              295
Tata              255
Renault           187
Volkswagen        171
Chevrolet         127
Skoda              93
Mercedes-Benz      79
Audi               73
Bmw                59
Nissan             53
Datsun             33
Other              22
Jeep               20
Fiat               18
Bajaj              16
Land               15
Force              14
Mercedes           13
Mitsubishi         12
Kia                12
Jaguar             10
Mg                  9
Volvo               9
Mini                8
Porsche             8
KIA                 7
BMW                 7
Ambassador          4
Ashok               4
MG                  3
```

Above fig shows total values of company column.

```
df['Name'].value_counts()
```

```
Maruti Suzuki          1364
Maruti Swift Dzire      154
Maruti Swift VDI        145
Toyota Innova           134
Honda City              116
                       ...
Nissan X-Trail            1
Renault Kwid RXL1.0       1
Mahindra Scorpio LX       1
Renault Duster RXS        1
ISUZU MU-7 High           1
Name: Name, Length: 507, dtype: int64
```

Above fig shows total values of name column.

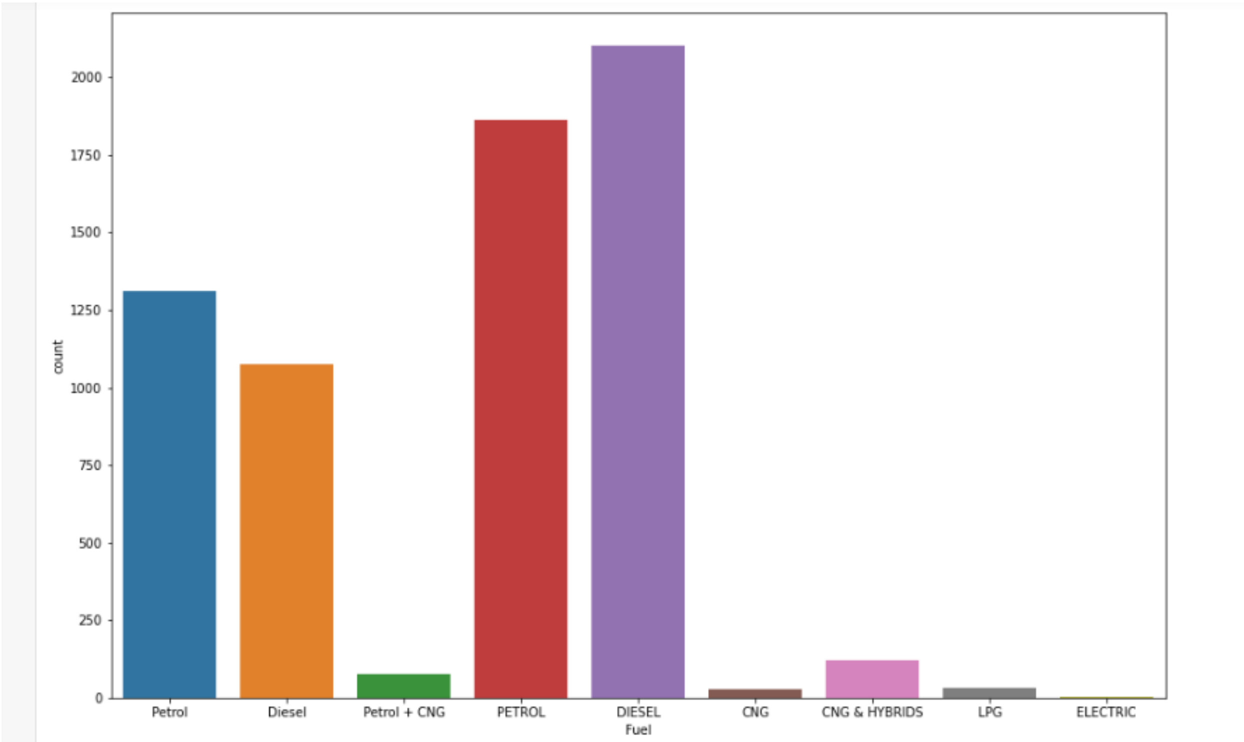
```
print(df['Fuel'].value_counts())
plt.figure(figsize=(15,10))
sns.countplot(df['Fuel'])
```

```
DIESEL            2102
PETROL            1864
Petrol            1309
Diesel            1076
CNG & HYBRIDS      119
Petrol + CNG        75
LPG                 31
CNG                 26
ELECTRIC             5
Name: Fuel, dtype: int64

<AxesSubplot:xlabel='Fuel', ylabel='count'>
```

```
df['Fuel'].value_counts()
plt.figure(figsize=(15,10))
sns.countplot(df['Fuel'])
```

```
<AxesSubplot:xlabel='Fuel', ylabel='count'>
```

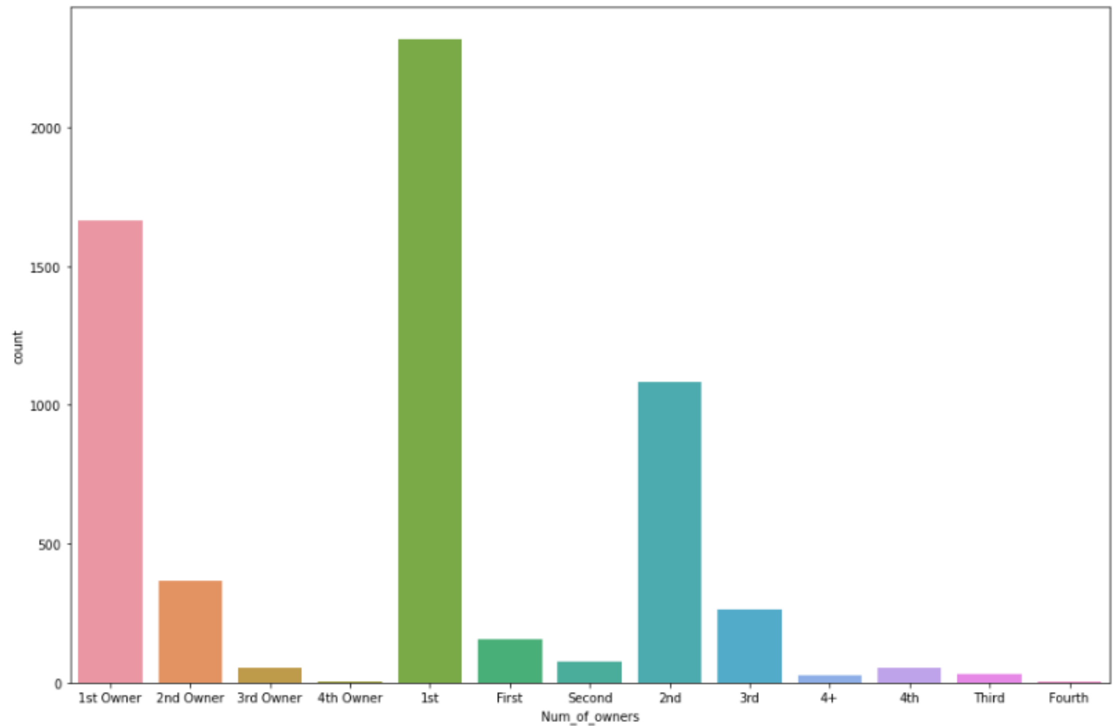



Above countplot  shows different features of fuel column.

```
print(df['Num_of_owners'].value_counts())
plt.figure(figsize=(15,10))
sns.countplot(df['Num_of_owners'])
```

```
1st          2317
1st Owner    1663
2nd          1083
2nd Owner     368
3rd           262
First         158
Second         75
3rd Owner      53
4th            52
Third          31
4+             28
4th Owner       6
Fourth          5
Name: Num_of_owners, dtype: int64
```

<AxesSubplot:xlabel='Num_of_owners', ylabel='count'>




Above countplot shows total values for num_of owners column.

```
:   1  df['Driven_kilometers'].value_counts()
```

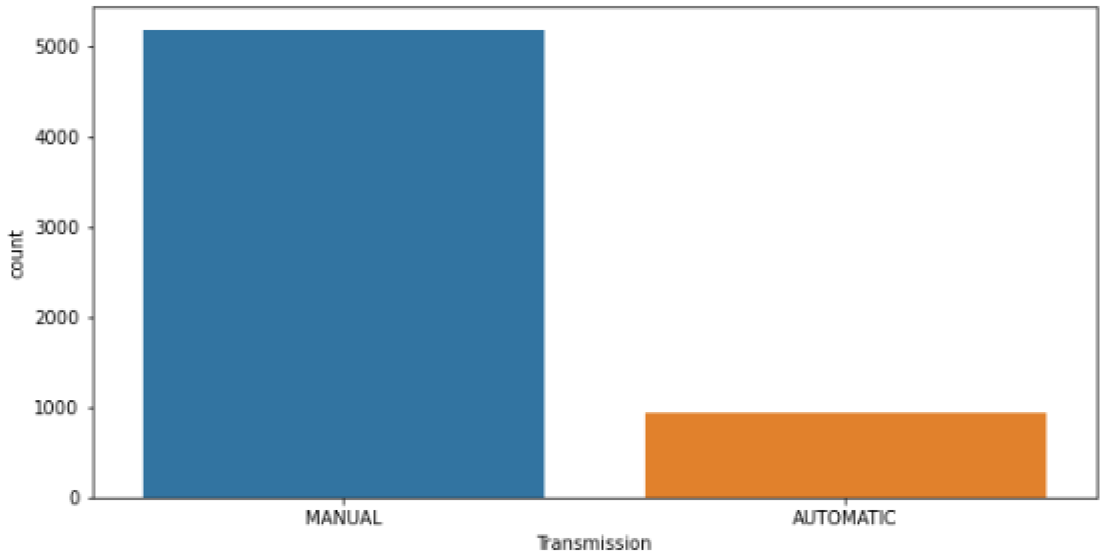
```
: 90,000 KM      51
  100,000 KM     49
  68000.0 KM     42
  65000.0 KM     41
  70000.0 KM     39
                 ..
  57100.0 KM      1
  51,392 km       1
  1,47,798 km     1
  95500.0 KM      1
  51500.0 KM      1
  Name: Driven_kilometers, Length: 3923, dtype: int64
```

Above fig shows total values for driven_kilometers column.

```
print(df['Transmission'].value_counts())
plt.figure(figsize=(10,5))
sns.countplot(df['Transmission'])
```

```
MANUAL       5184
AUTOMATIC     945
Name: Transmission, dtype: int64

<AxesSubplot:xlabel='Transmission', ylabel='count'>
```




Above fig shows values for transmission column.

```
df['Location'].value_counts()
```

```
DL-8C                          279
HR-26                          227
DL-3C                          210
DL-9C                          162
DL-2C                          132
                              ...
Bally Khal, Kolkata              1
Naraina, Delhi                   1
Kavumbhagam, Thiruvalla          1
Rohini Sector 18, Delhi          1
Military Station, Hisar          1
Name: Location, Length: 1970, dtype: int64
```

Above fig shows values for location column.

```
df['Year'].value_counts()
```

```
2016)     431
2012)     379
2017)     356
2015)     348
2013)     344
2014)     343
2018)     341
2015      324
2013      308
2011)     304
2014      297
2017      277
2018      251
2016      248
2019)     235
2010)     234
2012      230
2009)     172
2008)     158
2019      158
2007)     125
2020       96
2020)      95
2010       94
2006)      88
2011       87
2021)      57
2009       51
2005)      43
2004)      34
2003)      31
2008       24
2002)      22
2001)      22
```

Above fig shows different year values for year column.

```
df['Price1'].value_counts()
```

```
 4,50,000     68
 2,50,000     57
 6,50,000     55
 1,50,000     47
 3,25,000     45
              ..
4,29,899       1
 13,99,000     1
 83,000        1
3,90,599       1
4,06,599       1
Name: Price1, Length: 2706, dtype: int64
```

Above fig shows values for price column.

**Preprocessing data:**

First we have to change data type of price,driven_kilometers,year into numeric type. So following steps are followed.

Price column:

```
df['Price1']=df['Price1'].str.replace(",",'')
df['Price1']=df['Price1'].astype(int)
df['Price1']
```

```
0        200199
1        321599
2        808699
3        242299
4        276199
          ...
6707    4500000
6708     899000
6709     190000
6710    1799000
6711    1850000
Name: Price1, Length: 6692, dtype: int32
```

## Driven_kilometers column:

```
[ ]: df['Driven_kilometers']=df['Driven_kilometers'].str.split(" ").str.get(0).str.replace(",",'')
     df['Driven_kilometers']=df['Driven_kilometers'].str.split(".").str.get(0)
     df['Driven_kilometers']=df['Driven_kilometers'].fillna(0)
     df['Driven_kilometers']=df['Driven_kilometers'].astype(int)
     df['Driven_kilometers']
```

```
[ ]: 0          23
     1       12535
     2        2589
     3       40184
     4        9217
             ...
     6707    30000
     6708    61000
     6709    79000
     6710    28000
     6711    35000
     Name: Driven_kilometers, Length: 6712, dtype: int32
```

## Year column:

```
df['Year']=df['Year'].str.replace(")",'')
df['Year']=df['Year'].str.replace(".",'')
df['Year']=df['Year'].astype(int)
df['Total_years']=2021-df['Year']
df.drop("Year",axis=1,inplace=True)
df['Total_years']
```

```
0        7
1        7
2        0
3        8
4        6
        ..
6707     7
6708    12
6709     9
6710     5
6711     5
Name: Total_years, Length: 6692, dtype: int32
```

In year column, new column total year is calculated.

Num_of_owners column:

```
df['Num_of_owners']=df['Num_of_owners'].replace('1st',1)
df['Num_of_owners']=df['Num_of_owners'].replace('1st Owner',1)
df['Num_of_owners']=df['Num_of_owners'].replace('First',1)
df['Num_of_owners']=df['Num_of_owners'].replace('2nd',2)
df['Num_of_owners']=df['Num_of_owners'].replace('2nd Owner',2)
df['Num_of_owners']=df['Num_of_owners'].replace('Second',2)
df['Num_of_owners']=df['Num_of_owners'].replace('3rd',3)
df['Num_of_owners']=df['Num_of_owners'].replace('3rd Owner',3)
df['Num_of_owners']=df['Num_of_owners'].replace('Third',3)
df['Num_of_owners']=df['Num_of_owners'].replace('4th',4)
df['Num_of_owners']=df['Num_of_owners'].replace('4+',5)
df['Num_of_owners']=df['Num_of_owners'].replace('4th Owner',4)
df['Num_of_owners']=df['Num_of_owners'].replace('Fourth',4)
```

```
df['Num_of_owners']=df['Num_of_owners'].fillna(0)
```

```
df['Num_of_owners']=df['Num_of_owners'].astype(int)
df['Num_of_owners']
```

```
0       1
1       1
2       1
3       1
4       1
       ..
6707    3
6708    2
6709    1
6710    2
6711    1
Name: Num_of_owners, Length: 6712, dtype: int32
```

Above fig shows that how num_of_owners column is converted into numerical form.

## Filling missing values:

```
df['Fuel']=df['Fuel'].fillna(df['Fuel'].mode()[0])
df['Transmission']=df['Transmission'].fillna("no_info")
df['Location']=df['Location'].fillna("no_info")
```

```
df.isnull().sum()
```

```
Unnamed: 0           0
Fuel                 0
Driven_kilometers    0
Num_of_owners        0
Transmission         0
Location             0
Name                 0
Company              0
Price1               0
Total_years          0
dtype: int64
```

Missing values of Fuel column is filled with mode of column, and location and transmission column's missing values are filled with 'no info' values.

Now there is no missing values in data set.

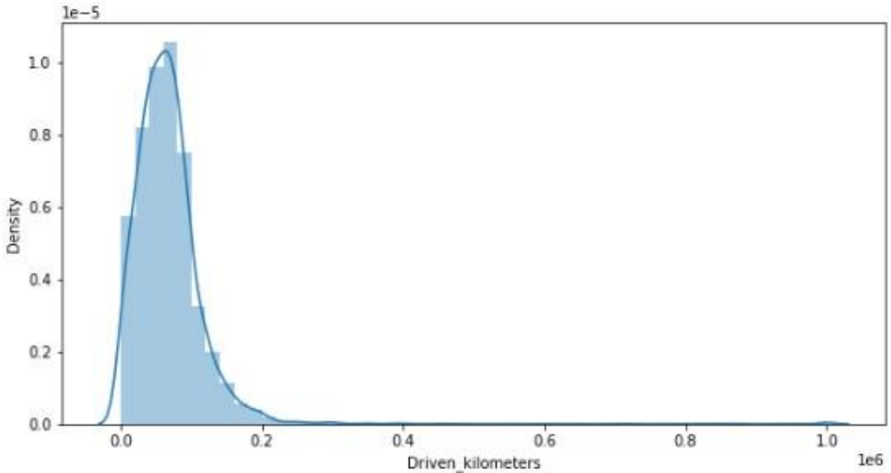
## Statistical summary:

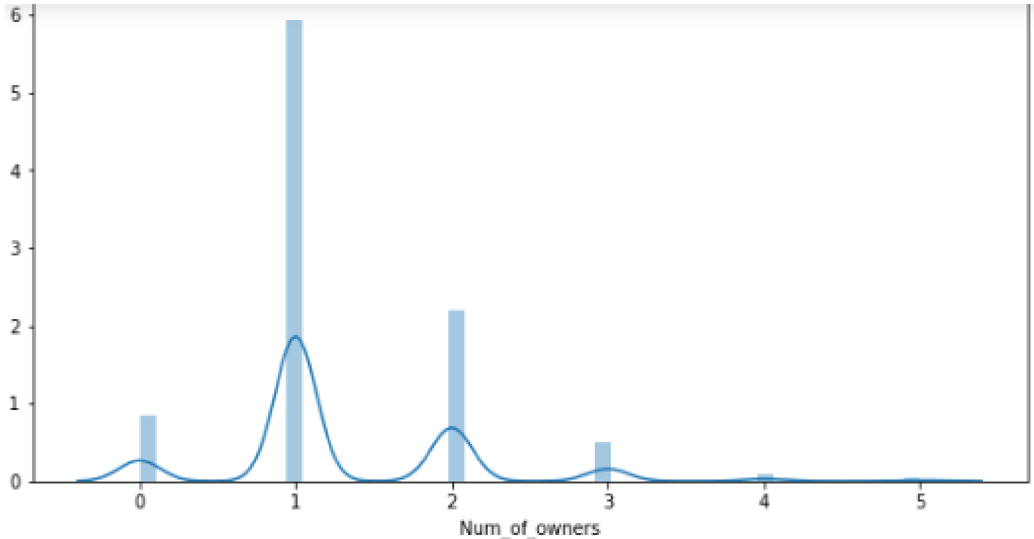
```
df.describe()
```

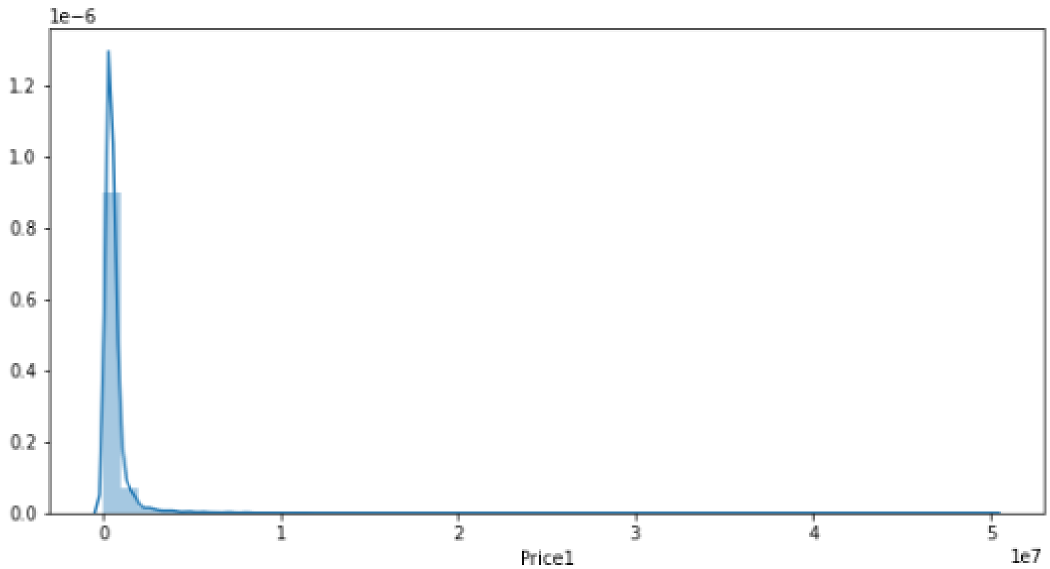
| | Unnamed: 0 | Driven_kilometers | Num_of_owners | Price1 | Total_years |
|---|---|---|---|---|---|
| count | 6692.000000 | 6692.000000 | 6692.000000 | 6.692000e+03 | 6692.000000 |
| mean | 3352.645846 | 67782.511955 | 1.288105 | 5.884777e+05 | -6.377316 |
| std | 1939.191426 | 58408.572353 | 0.769363 | 9.602317e+05 | 495.499316 |
| min | 0.000000 | 0.000000 | 0.000000 | 1.500000e+04 | -18179.000000 |
| 25% | 1672.750000 | 37000.000000 | 1.000000 | 2.650000e+05 | 4.000000 |
| 50% | 3349.500000 | 61866.000000 | 1.000000 | 4.097990e+05 | 7.000000 |
| 75% | 5035.250000 | 86554.250000 | 2.000000 | 6.423990e+05 | 9.000000 |
| max | 6711.000000 | 999999.000000 | 5.000000 | 5.000000e+07 | 63.000000 |

```
ncols=['Driven_kilometers','Num_of_owners','Price1','Total_years']
for i in df[ncols]:
    plt.figure(figsize=(10,5))
    sns.distplot(df[i])

```






Above distplot shows distribution of the data. As the data is originally contains object type so we can’t do some transformation on data set.

**Encoder:**

Encoding: First we have to encode the categorical data into numerical data. There are different techniques of encoding:

- One Hot Encoder: Encode categorical integer features using a onehot aka one-of-K scheme. The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature.
- Label Encoder: Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important preprocessing step for the structured dataset in supervised learning
- Ordinal Encoder: In ordinal encoding, each unique categoryvalue is assigned an integer value. For example, “red” is 1,“green” is 2, and “blue” is 3. This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

In this project some columns are encoded using get dummies method of one hot encoder and some columns are encoded using label encoder.

```
dummies=pd.get_dummies(df[['Fuel','Transmission','Company']])
dummies.head()
```

| | Fuel_CNG | Fuel_CNG & HYBRIDS | Fuel_DIESEL | Fuel_Diesel | Fuel_ELECTRIC | Fuel_LPG | Fuel_PETROL | Fuel_Petrol | Fuel_Petrol + CNG | Transmission_AUTOMATIC | ... | Company |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |

5 rows × 54 columns

Above fig shows that Fuel, Transmission and company column is encoded using get dummies method of one hot encoder.

```
le=LabelEncoder()

df['Name']=le.fit_transform(df['Name'])
```

Name column is encoded using label encoder.

Now delete fuel, transmission and company column and concat dummies columns into data set.

Below fig shows this.

```
df.drop(['Fuel','Transmission','Company'],axis=1,inplace=True)
```

```
df2=pd.concat([df,dummies],axis=1)
```

```
df2.head()
```

| | Unnamed: 0 | Driven_kilometers | Num_of_owners | Location | Name | Price1 | Total_years | Fuel_CNG | Fuel_CNG & HYBRIDS | Fuel_DIESEL | ... | Company_Nissan | Company_Ot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 23 | 1 | DL-4C | 301 | 200199 | 7 | 0 | 0 | 0 | ... | 0 | |
| 1 | 1 | 12535 | 1 | DL-12 | 259 | 321599 | 7 | 0 | 0 | 0 | ... | 0 | |
| 2 | 2 | 2589 | 1 | UP-32 | 160 | 808699 | 0 | 0 | 0 | 0 | ... | 0 | |
| 3 | 3 | 40184 | 1 | DL-8C | 260 | 242299 | 8 | 0 | 0 | 0 | ... | 0 | |
| 4 | 4 | 9217 | 1 | DL-8C | 259 | 276199 | 6 | 0 | 0 | 0 | ... | 0 | |

5 rows × 61 columns

## Dropping columns:

```
df2.drop("Location",axis=1,inplace=True)
```

```
df2.drop("Unnamed: 0",axis=1,inplace=True)
```

Dropped columns which are not so much important.

Separate data into input variables and target variable;

```
x=df2.drop(["Price1"],axis=1)
y=df2['Price1']
```

## Standard Scaler:

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.

This can be thought of as subtracting the mean value or centering the data.

Like normalization, standardization can be useful, and even required in some machine learning algorithms where data has input values with differing scales.

```python
#standard scaler
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
cols=['Driven_kilometers','Name']
x[cols]=st.fit_transform(x[cols])

```

```python
x_1=pd.DataFrame(x,columns=x.columns)
```

```python
x_1.head()
```

| | Driven_kilometers | Num_of_owners | Name | Total_years | Fuel_CNG | Fuel_CNG & HYBRIDS | Fuel_DIESEL | Fuel_Diesel | Fuel_ELECTRIC | Fuel_LPG | ... | Company_Nissan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.160182 | 1 | 0.331323 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 |
| 1 | -0.945951 | 1 | -0.007760 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 |
| 2 | -1.116247 | 1 | -0.807026 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 |
| 3 | -0.472543 | 1 | 0.000314 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 |
| 4 | -1.002762 | 1 | -0.007760 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 |

5 rows × 58 columns

Above fig shows that driven_kilometers and name column is standardized using standard scaler.

## Split data for training and testing

```python
# split training data for training and testing
x_train,x_test,y_train,y_test=train_test_split(x_1,y,test_size=.25,random_state=219)
print("x_train shape",x_train.shape)
print("x_test shape",x_test.shape)
print("y_train shape",y_train.shape)
print("y_test shape",y_test.shape)
```

```
x_train shape (5019, 58)
x_test shape (1673, 58)
y_train shape (5019,)
y_test shape (1673,)
```

Above fig shows that data is split using train_test_split for training and testing.

Now create instance of modules. As this is regression type problem so we have to import regression algorithms.

```
lr=LinearRegression()
dtr=DecisionTreeRegressor()
rf=RandomForestRegressor()
svr=SVR()
l1=Lasso(alpha=0.001)
r1=Ridge(alpha=0.001)

```

## Fit and predict:

Now fit data into model and predict the output.

```
#fit data  and predict
list1=[lr,dtr,rf,svr,l1,r1]
for i in list1:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    print("accuracy_scores",i)
    print("r2_score",r2_score(y_test,pred))
    print("mean_squared_error",mean_squared_error(y_test,pred))
    print("mean_absolute_error",mean_absolute_error(y_test,pred))

```

```
accuracy_scores LinearRegression()
r2_score 0.4674687461890442
mean_squared_error 231764108396.26334
mean_absolute_error 258732.22603485011
accuracy_scores DecisionTreeRegressor()
r2_score 0.7012651817148703
mean_squared_error 130013043011.65211
mean_absolute_error 147055.79796772264
accuracy_scores RandomForestRegressor()
r2_score 0.8553926996841921
mean_squared_error 62934863982.99051
mean_absolute_error 121308.64814888041
accuracy_scores SVR()
r2_score -0.05078857103084777
mean_squared_error 457316025181.87787
mean_absolute_error 310615.73047899746
accuracy_scores Lasso(alpha=0.001)
r2_score 0.4656743997152657
mean_squared_error 232545029905.88965
mean_absolute_error 258448.13751463423
accuracy_scores Ridge(alpha=0.001)
r2_score 0.4674724764481426
mean_squared_error 231762484941.9255
mean_absolute_error 258731.36615293552
```

By observing metrics we can say that r2_score of randomForestRegressor is high.

## HyperParameterTunning

```python
from sklearn.ensemble import RandomForestRegressor
rf1 = RandomForestRegressor()

from sklearn.model_selection import GridSearchCV
param_grid = {
            "n_estimators"      : [10,20,30],
            "max_features"      : ["auto", "sqrt", "log2"],
            "min_samples_split" : [2,4,8],
            "bootstrap": [True, False],
            }

grid = GridSearchCV(estimator=rf, param_grid=param_grid, n_jobs=-1, cv=5)

grid.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 4, 8],
                         'n_estimators': [10, 20, 30]})
```

```python
grid.best_params_
```

```
{'bootstrap': False,
 'max_features': 'sqrt',
 'min_samples_split': 4,
 'n_estimators': 20}
```

We can tune different parameters of model so we can improve model’s score.

```python
rf1=RandomForestRegressor(bootstrap=False,max_features= 'sqrt',min_samples_split=4,n_estimators=20)
rf1.fit(x_train,y_train)
rpred=rf1.predict(x_test)
cv3=cross_val_score(rf1,x_train,y_train,cv=5)
print("score",cv3)
print("cross score mean value",cv3.mean())
print('mean squared error',mean_squared_error(rpred,y_test))
print(r2_score(y_test,rpred))
```

```
score [0.65477603 0.61695564 0.56479431 0.11027668 0.50882151]
cross score mean value 0.491124834827942
mean squared error 109943330852.28693
0.7473799535568422
```

After applying best parameters we got by tunning, r2_score is not increased, so we have to select model which gave high score.

So by observing first scores of RandomForestRegressor before tuning, we get high score, so that model is best.

Now let's create object file.

```
#creating object file
import joblib
```

```
joblib.dump(rf,"car_prediction.obj")
```

```
['car_prediction.obj']
```

```
file1=joblib.load("car_prediction.obj")
```

```
file1.predict(x_test)
```

```
array([1704374.82,  383318.  , 1208429.99, ...,  357743.05,  386209.51,
        588725.  ])
```

# Conclusion

The increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. The proposed system will help to determine the accurate price of used car price prediction.

In this project different regression algorithms are used, and Random Forest Regressor give high score so selected that one.

# Future scope

We may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset.