Used Car Price Prediction

So called Second hand's car have a huge market base. Many consider to buy a Used Car intsead of buying of new one, as it's is feasible and a better investment.

The main reason for this huge market is that when you buy a New Car and sale it just another day without any default on it, the price of car reduces by 30%.

There are also many frauds in the market who not only sale wrong but also they could mislead to wrong price.

So we have built a model that predicts the price of any used car.

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

```
train_data = pd.read_csv('/content/drive/MyDrive/AIDS MP/train-data.csv')
test_data = pd.read_csv('/content/drive/MyDrive/AIDS MP/test-data.csv')
```

The info() method prints information about the DataFrame.

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6019 entries, 0 to 6018
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         6019 non-null   int64  
 1   Name               6019 non-null   object 
 2   Location           6019 non-null   object 
 3   Year               6019 non-null   int64  
 4   Kilometers_Driven  6019 non-null   int64  
 5   Fuel_Type          6019 non-null   object 
 6   Transmission       6019 non-null   object 
 7   Owner_Type         6019 non-null   object 
 8   Mileage            6017 non-null   object 
 9   Engine             5983 non-null   object 
 10  Power              5983 non-null   object 
 11  Seats              5977 non-null   float64
 12  New_Price          824 non-null    object 
```

```
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 658.5+ KB
```

The head() function is used to get the first n rows.

```
train_data.head()
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual

The tail() method returns a specified number of last rows.

```
train_data.tail()
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmissio
6014	6014	Maruti Swift VDI	Delhi	2014	27365	Diesel	Manua
6015	6015	Hyundai Xcent 1.1 CRDi S	Jaipur	2015	100000	Diesel	Manua
		Mahindra					

The iloc() function in python is defined in the Pandas module that helps us to select a specific row or column from the data set.

```
train_data = train_data.iloc[:,1:]
train_data.head()
```

The describe() function is used to get a descriptive statistics summary of a given dataframe.

```
train_data.describe()
```

	Year	Kilometers_Driven	Seats	Price
count	6019.000000	6.019000e+03	5977.000000	6019.000000
mean	2013.358199	5.873838e+04	5.278735	9.479468
std	3.269742	9.126884e+04	0.808840	11.187917
min	1998.000000	1.710000e+02	0.000000	0.440000
25%	2011.000000	3.400000e+04	5.000000	3.500000
50%	2014.000000	5.300000e+04	5.000000	5.640000
75%	2016.000000	7.300000e+04	5.000000	9.950000
max	2019.000000	6.500000e+06	10.000000	160.000000

The shape of a DataFrame is a tuple of array dimensions that tells the number of rows and columns of a given DataFrame.

```
train_data.shape
```

```
(6019, 13)
```

The value_counts() return a Series containing counts of unique values.

```
train_data['Kilometers_Driven'].value_counts()
```

```
60000    82
45000    70
65000    68
50000    61
55000    60
         ..
28937     1
82085     1
68465     1
63854     1
27365     1
Name: Kilometers_Driven, Length: 3093, dtype: int64
```

The unique() function returns the unique values present in a dataset.

```
# Looking at the unique values of Categorical Features
print(train_data['Location'].unique())
print(train_data['Fuel_Type'].unique())
```

```
print(train_data['Transmission'].unique())
print(train_data['Owner_Type'].unique())

#Rest Feature are worked for Feature Engineering
```

```
['Mumbai' 'Pune' 'Chennai' 'Coimbatore' 'Hyderabad' 'Jaipur' 'Kochi'
 'Kolkata' 'Delhi' 'Bangalore' 'Ahmedabad']
['CNG' 'Diesel' 'Petrol' 'LPG' 'Electric']
['Manual' 'Automatic']
['First' 'Second' 'Fourth & Above' 'Third']
```

The isnull().sum() returns the sum of NULL values for individual colum

```
train_data.isnull().sum()
```

```
Name                   0
Location               0
Year                   0
Kilometers_Driven      0
Fuel_Type              0
Transmission           0
Owner_Type             0
Mileage                2
Engine                36
Power                 36
Seats                 42
New_Price           5195
Price                  0
dtype: int64
```

Let's Drop sum Rows which contains NULL values.

NOTE: We are ignoring New_Price Column as it contains many cells which contains NULL value which will drastically shrink our train dataset.

The notna() function detects existing/ non-missing values in the dataframe.

```
print("Shape of train data Before dropping any Row: ",train_data.shape)
train_data = train_data[train_data['Mileage'].notna()]
print("Shape of train data After dropping Rows with NULL values in Mileage: ",train_data.s
train_data = train_data[train_data['Engine'].notna()]
print("Shape of train data After dropping Rows with NULL values in Engine : ",train_data.s
train_data = train_data[train_data['Power'].notna()]
print("Shape of train data After dropping Rows with NULL values in Power  : ",train_data.s
train_data = train_data[train_data['Seats'].notna()]
print("Shape of train data After dropping Rows with NULL values in Seats  : ",train_data.s
```

```
Shape of train data Before dropping any Row:  (6019, 13)
Shape of train data After dropping Rows with NULL values in Mileage:  (6017, 13)
Shape of train data After dropping Rows with NULL values in Engine :  (5981, 13)
Shape of train data After dropping Rows with NULL values in Power  :  (5981, 13)
Shape of train data After dropping Rows with NULL values in Seats  :  (5975, 13)
```

Now here in total we have 5975 Rows to work forward with. We have droped 44 rows.

Now after using .notna() function, we have many absent indexes (Eg: If row no 47 was droped then after 46 we have 48 index), so we will reset the index and droping the present index.

```
train_data = train_data.reset_index(drop=True)
```

```
train_data.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	Firs
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	Firs

Feautre Engineering

There are many different data which could be extarcted from present. And, that's where Feature Engineering comes.

Following block of code creates a new alternate column for every existing column where in we are spliting the data in the cell in order to get only the first integer part.

```
for i in range(train_data.shape[0]):
    train_data.at[i, 'Company'] = train_data['Name'][i].split()[0]
    train_data.at[i, 'Mileage(km/kg)'] = train_data['Mileage'][i].split()[0]
    train_data.at[i, 'Engine(CC)'] = train_data['Engine'][i].split()[0]
    train_data.at[i, 'Power(bhp)'] = train_data['Power'][i].split()[0]
```

```
train_data.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	Firs
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	Firs

The astype() function is used to convert the column from string/int to float

```
train_data['Mileage(km/kg)'] = train_data['Mileage(km/kg)'].astype(float)
train_data['Engine(CC)'] = train_data['Engine(CC)'].astype(float)
```

Audi A4

At this point when we tried to change Power(bhp) to float an error occured (Can't convert str to float : null).

This is because some cell where having values: 'null bhp'

```
train_data['Power'][76]
```

```
'null bhp'
```

Following block of code returns us with the total count of the NULL-value cells along with their indices

```
count = 0
position = []
for i in range(train_data.shape[0]):
    if train_data['Power(bhp)'][i]=='null':

        count = count + 1
        position.append(i)

print(count)
print(position)
```

```
103
[76, 79, 89, 120, 143, 225, 242, 259, 304, 305, 383, 421, 425, 440, 469, 572, 628, 64
```

Dropping the NULL-value cells of power(bhp) column

```
train_data = train_data.drop(train_data.index[position])
train_data = train_data.reset_index(drop=True)
```

The new shape of training data is:

```
train_data.shape
```

```
(5872, 17)
```

Now the power column is successfully converted to float

```
train_data['Power(bhp)'] = train_data['Power(bhp)'].astype(float)
```

```
train_data.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	Firs
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	Firs
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	Firs
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	Firs
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Seconc

Here we are splitting the cell data of New_Price column and storing the splitted integer data into the new column named New_car_Price

```
for i in range(train_data.shape[0]):
    if pd.isnull(train_data.loc[i,'New_Price']) == False:
        train_data.at[i,'New_car_Price'] = train_data['New_Price'][i].split()[0]
```

```
train_data.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	Firs
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	Firs
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	Firs

Converting New_car_Price column from string to float datatype

```
train_data['New_car_Price'] = train_data['New_car_Price'].astype(float)
```

Now,

Let's delete all the redudant features(Features that are not going to help us further).

```
train_data.drop(["Name"],axis=1,inplace=True)
train_data.drop(["Mileage"],axis=1,inplace=True)
train_data.drop(["Engine"],axis=1,inplace=True)
train_data.drop(["Power"],axis=1,inplace=True)
train_data.drop(["New_Price"],axis=1,inplace=True)
```

DATA VISUALIZATION

Data visualization is the best way to find out how a data looks like.

The info() method prints information about the DataFrame

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5872 entries, 0 to 5871
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Location           5872 non-null   object
 1   Year               5872 non-null   int64
 2   Kilometers_Driven  5872 non-null   int64
 3   Fuel_Type          5872 non-null   object
 4   Transmission       5872 non-null   object
 5   Owner_Type         5872 non-null   object
 6   Seats              5872 non-null   float64
 7   Price              5872 non-null   float64
 8   Company            5872 non-null   object
 9   Mileage(km/kg)     5872 non-null   float64
 10  Engine(CC)         5872 non-null   float64
 11  Power(bhp)         5872 non-null   float64
```

```
 12  New_car_Price      823 non-null    float64
dtypes: float64(6), int64(2), object(5)
memory usage: 596.5+ KB
```

The describe() method returns description of the data in the DataFrame

```
train_data['New_car_Price'].describe()
```

```
count    823.000000
mean      20.328906
std       20.209032
min        1.000000
25%        7.840000
50%       11.390000
75%       24.010000
max       99.920000
Name: New_car_Price, dtype: float64
```

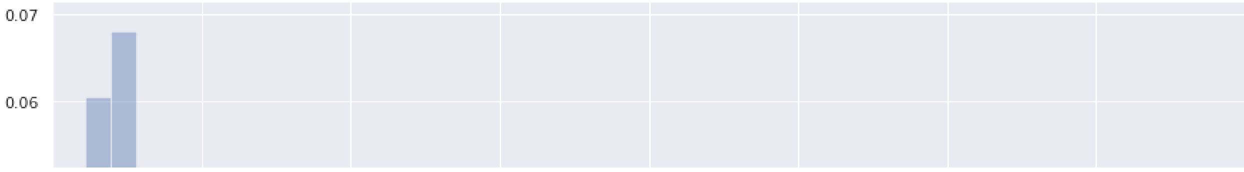
Price

First let's have a look over our target column

The distplot() function is used to plot the distplot. The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution.

```
f, ax = plt.subplots(figsize=(15,8))
sns.distplot(train_data['New_car_Price'])
plt.xlim([0,160])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
(0.0, 160.0)
```

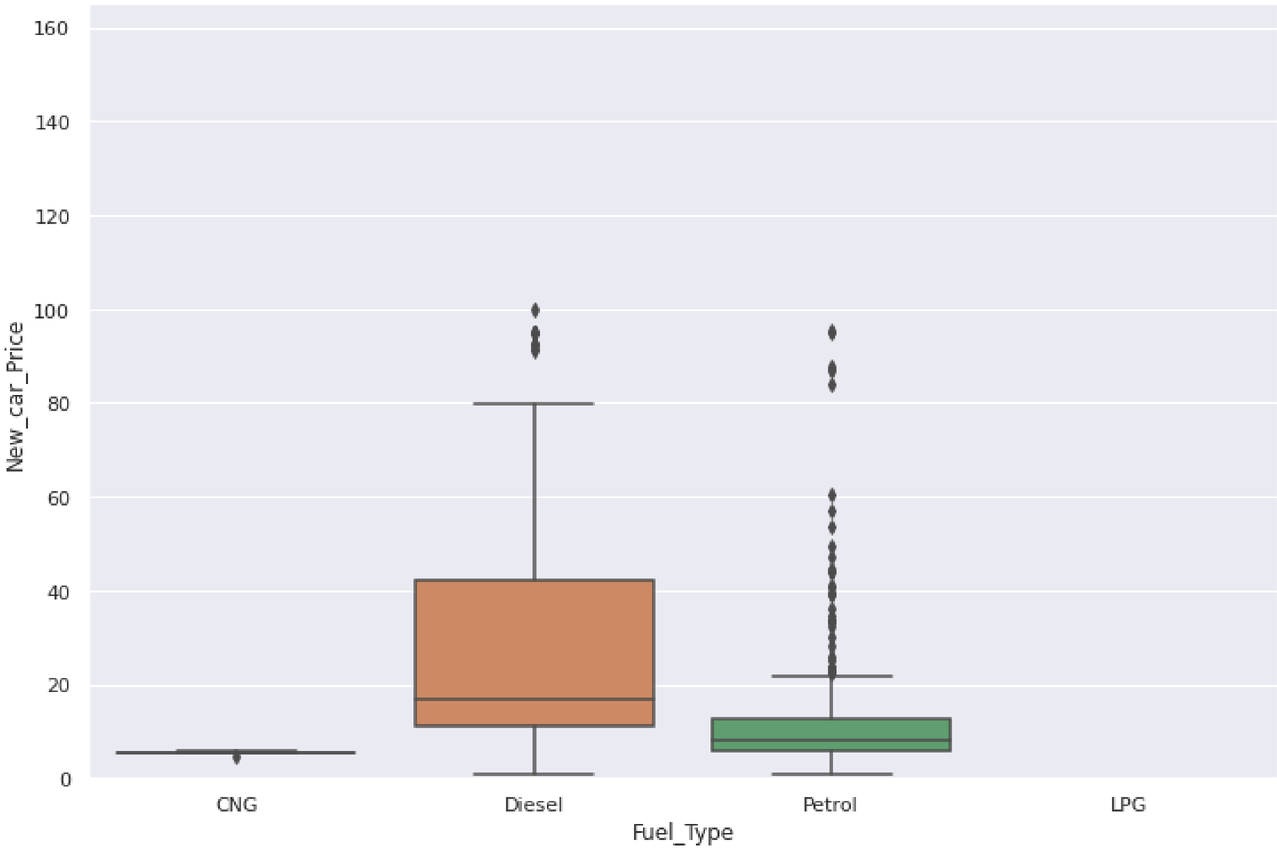


Fuel Type

```
train_data['Fuel_Type'].describe()
```

```
count       5872
unique         4
top       Diesel
freq        3152
Name: Fuel_Type, dtype: object
```

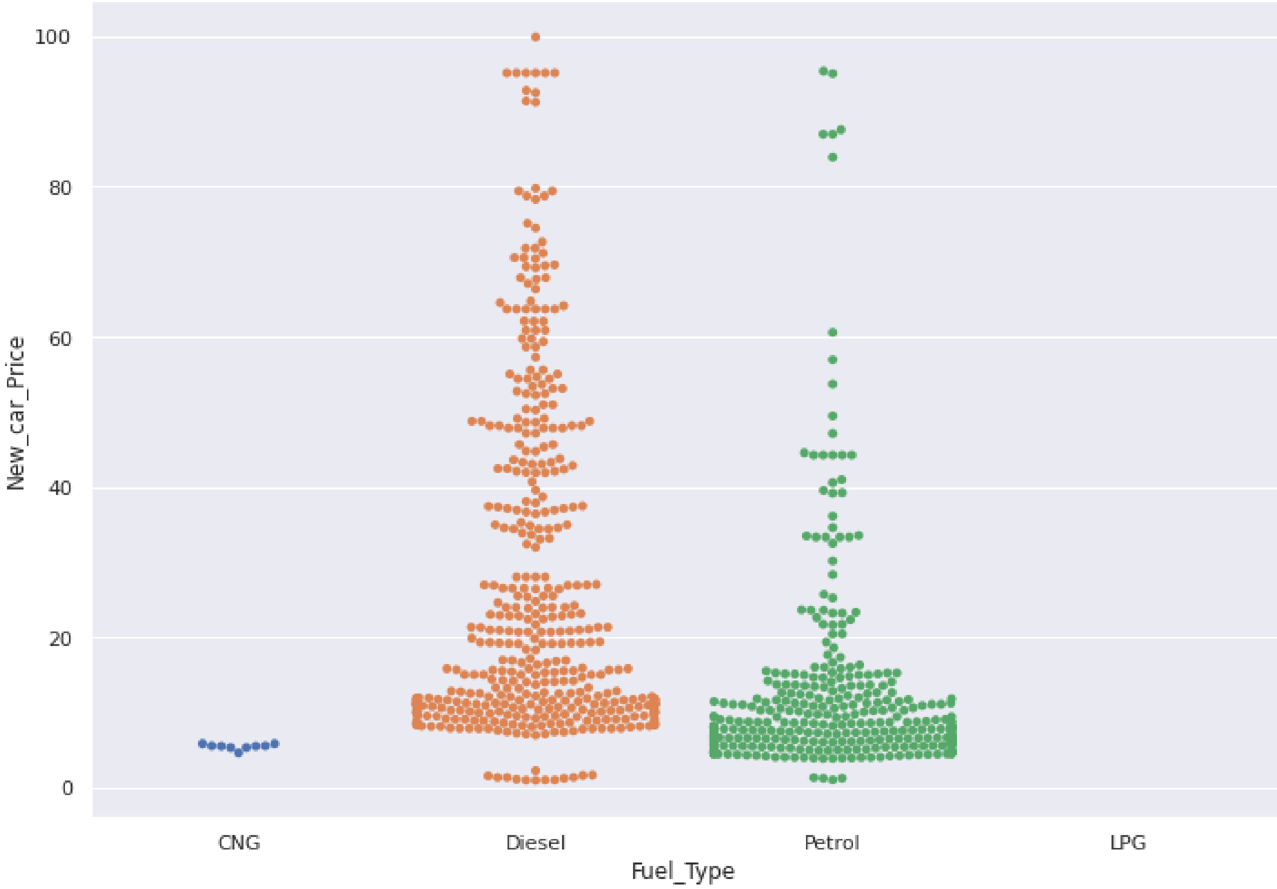
```
var = 'Fuel_Type'
data = pd.concat([train_data['New_car_Price'], train_data[var]], axis=1)
f, ax = plt.subplots(figsize=(12, 8))
fig = sns.boxplot(x=var, y="New_car_Price", data=data)
fig.axis(ymin=0, ymax=165);
```



Diseal car would cost followed Petrol.

```
var = 'Fuel_Type'
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.swarmplot(x = var, y ='New_car_Price', data = train_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f857aacc7d0>
```

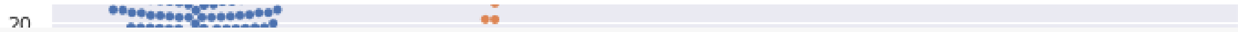


Owner Type

```
var = 'Owner_Type'
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.swarmplot(x = var, y ='New_car_Price', data = train_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f857aae1a50>
```

Company




```
var = "Company"
plt.figure(figsize=(20, 10))
sns.catplot(x=var, kind="count", palette="ch:.25", height=8, aspect=2, data=train_data);
plt.xticks(rotation=90);
```

```
<Figure size 1440x720 with 0 Axes>
```

Cleary Maruti is most comman brand followed by Hyundai

Working with Categorical Data

As for now we have left with only 5 categorical features:

Location Fuel_Type Transmission Owner_Type Company For hadeling categorical data. We modtly use these 2 path:

OneHotEncoder LabelEncoder Where OneHotEncoder is used where data are not in any order and LabelEncoder when data is in order.

So, for each Features we will use plots to find out what to be used there.

Working for Location

```
var = 'Location'
train_data[var].value_counts()
```

```
Mumbai        775
Hyderabad     718
Kochi         645
Coimbatore    629
Pune          594
Delhi         545
Kolkata       521
Chennai       476
Jaipur        402
Bangalore     347
Ahmedabad     220
Name: Location, dtype: int64
```

From above values, we could judge that Mubmai has most number of cars to be sold followed by others.

We will be using One-hot-encoding here

```
Location = train_data[[var]]
Location = pd.get_dummies(Location,drop_first=True)
Location.head()
```

	Location_Bangalore	Location_Chennai	Location_Coimbatore	Location_Delhi	Locati
0	0	0	0	0	
1	0	0	0	0	
3	0	1	0	0	

Working for Fuel_Type

```
var = 'Fuel_Type'
train_data[var].value_counts()
```

```
Diesel    3152
Petrol    2655
CNG         55
LPG         10
Name: Fuel_Type, dtype: int64
```

Again we will be using One-hot-encoding

```
Fuel_t = train_data[[var]]
Fuel_t = pd.get_dummies(Fuel_t,drop_first=True)
Fuel_t.head()
```

	Fuel_Type_Diesel	Fuel_Type_LPG	Fuel_Type_Petrol
0	0	0	0
1	1	0	0
2	0	0	1
3	1	0	0
4	1	0	0

Working with Transmission

```
var = 'Transmission'
train_data[var].value_counts()
```

```
Manual       4170
Automatic    1702
Name: Transmission, dtype: int64
```

No, order so One-hot-encoding

```
Transmission = train_data[[var]]
Transmission = pd.get_dummies(Transmission,drop_first=True)
Transmission.head()
```

	Transmission_Manual
0	1
1	1
2	1
3	1
4	0

Working with Owner_Type

```
var = 'Owner_Type'
train_data[var].value_counts()
```

```
First             4839
Second             925
Third              101
Fourth & Above       7
Name: Owner_Type, dtype: int64
```

As Owner_Type column has ordered data so we will be using Label Encoding Finally

```
train_data.replace({"First":1,"Second":2,"Third": 3,"Fourth & Above":4},inplace=True)
train_data.head()
```

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Seats	P
0	Mumbai	2010	72000	CNG	Manual	1	5.0	
1	Pune	2015	41000	Diesel	Manual	1	5.0	
2	Chennai	2011	46000	Petrol	Manual	1	5.0	
3	Chennai	2012	87000	Diesel	Manual	1	7.0	
4	Coimbatore	2013	40670	Diesel	Automatic	2	5.0	

Working with Company

```
var = 'Company'
train_data[var].value_counts()
```

```
Maruti           1175
Hyundai          1058
Honda             600
Toyota            394
Mercedes-Benz     316
Volkswagen        314
Ford              294
Mahindra          268
BMW               262
Audi              235
```

```
Tata             183
Skoda            172
Renault          145
Chevrolet        120
Nissan            89
Land              57
Jaguar            40
Mitsubishi        27
Mini              26
Fiat              23
Volvo             21
Porsche           16
Jeep              15
Datsun            13
Force              3
ISUZU              2
Ambassador         1
Isuzu              1
Bentley            1
Lamborghini        1
Name: Company, dtype: int64
```

A lot of variation so let's drop them

```
train_data.drop(["Company"],axis=1,inplace=True)
```

```
final_train= pd.concat([train_data,Location,Fuel_t,Transmission],axis=1)
final_train.head()
```

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Seats	P
0	Mumbai	2010	72000	CNG	Manual	1	5.0	
1	Pune	2015	41000	Diesel	Manual	1	5.0	
2	Chennai	2011	46000	Petrol	Manual	1	5.0	
3	Chennai	2012	87000	Diesel	Manual	1	7.0	
4	Coimbatore	2013	40670	Diesel	Automatic	2	5.0	

5 rows × 26 columns

```
final_train.drop(["Location","Fuel_Type","Transmission","New_car_Price"],axis=1,inplace=Tr
final_train.head()
```

	Year	Kilometers_Driven	Owner_Type	Seats	Price	Mileage(km/kg)	Engine(CC)	Pow
0	2010	72000	1	5.0	1.75	26.60	998.0	
1	2015	41000	1	5.0	12.50	19.67	1582.0	

```
final_train.shape
```

```
(5872, 22)
```

4	2013	40670	2	5.0	17.74	15.20	1968.0	

We are Done with Training data, so now work on Test Data

Prepare Test Data

```
test_data.head()
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	O
0	0	Maruti Alto K10 LXI CNG	Delhi	2014	40929	CNG	Manual	
1	1	Maruti Alto 800 2016	Coimbatore	2013	54493	Petrol	Manual	

We will have to prepare this test data with performing all the steps agaon for test data

```
test_data = test_data.iloc[:,1:]

print("Shape of test data Before dropping any Row: ",train_data.shape)
test_data = test_data[test_data['Mileage'].notna()]
print("Shape of test data After dropping Rows with NULL values in Mileage: ",test_data.sha
test_data = test_data[test_data['Engine'].notna()]
print("Shape of test data After dropping Rows with NULL values in Engine : ",test_data.sha
test_data = test_data[test_data['Power'].notna()]
print("Shape of test data After dropping Rows with NULL values in Power  : ",test_data.sha
test_data = test_data[test_data['Seats'].notna()]
print("Shape of test data After dropping Rows with NULL values in Seats  : ",test_data.sha
print('Droping null done')

test_data = test_data.reset_index(drop=True)
print('Index reset done')

for i in range(test_data.shape[0]):
    test_data.at[i, 'Mileage(km/kg)'] = test_data['Mileage'][i].split()[0]
    test_data.at[i, 'Engine(CC)'] = test_data['Engine'][i].split()[0]
    test_data.at[i, 'Power(bhp)'] = test_data['Power'][i].split()[0]
print('Split Done')

test_data['Mileage(km/kg)'] = test_data['Mileage(km/kg)'].astype(float)
```

```
test_data['Engine(CC)'] = test_data['Engine(CC)'].astype(float)
print('casting 1 Done')

position = []
for i in range(test_data.shape[0]):
    if test_data['Power(bhp)'][i]=='null':
        position.append(i)

test_data = test_data.drop(test_data.index[position])
test_data = test_data.reset_index(drop=True)

test_data['Power(bhp)'] = test_data['Power(bhp)'].astype(float)
print('casting 2 Done')

for i in range(test_data.shape[0]):
    if pd.isnull(test_data.loc[i,'New_Price']) == False:
        test_data.at[i,'New_car_Price'] = test_data['New_Price'][i].split()[0]

test_data['New_car_Price'] = test_data['New_car_Price'].astype(float)

test_data.drop(["Name"],axis=1,inplace=True)
test_data.drop(["Mileage"],axis=1,inplace=True)
test_data.drop(["Engine"],axis=1,inplace=True)
test_data.drop(["Power"],axis=1,inplace=True)
test_data.drop(["New_Price"],axis=1,inplace=True)

var = 'Location'
Location = test_data[[var]]
Location = pd.get_dummies(Location,drop_first=True)
Location.head()

var = 'Fuel_Type'
Fuel_t = test_data[[var]]
Fuel_t = pd.get_dummies(Fuel_t,drop_first=True)
Fuel_t.head()

var = 'Transmission'
Transmission = test_data[[var]]
Transmission = pd.get_dummies(Transmission,drop_first=True)
Transmission.head()

test_data.replace({"First":1,"Second":2,"Third": 3,"Fourth & Above":4},inplace=True)
test_data.head()

final_test= pd.concat([test_data,Location,Fuel_t,Transmission],axis=1)
final_test.head()

final_test.drop(["Location","Fuel_Type","Transmission","New_car_Price"],axis=1,inplace=Tru
final_test.head()

print("Final Test Size: ",final_test.shape)
```

```
Shape of test data Before dropping any Row:  (5872, 12)
Shape of test data After dropping Rows with NULL values in Mileage:  (1234, 12)
Shape of test data After dropping Rows with NULL values in Engine :  (1224, 12)
```

```
Shape of test data After dropping Rows with NULL values in Power  :  (1224, 12)
Shape of test data After dropping Rows with NULL values in Seats  :  (1223, 12)
Droping null done
Index reset done
Split Done
casting 1 Done
casting 2 Done
Final Test Size:  (1201, 21)
```

```
final_test.head()
```

	Year	Kilometers_Driven	Owner_Type	Seats	Mileage(km/kg)	Engine(CC)	Power(bhp
0	2014	40929	1	4.0	32.26	998.0	58.2(
1	2013	54493	2	5.0	24.70	796.0	47.3(
2	2017	34000	1	7.0	13.68	2393.0	147.8(
3	2014	29000	1	5.0	18.50	1197.0	82.8!
4	2016	85609	2	7.0	16.00	2179.0	140.0(

5 rows × 21 columns

Final Features Selection

As our train and test data are ready so now we have to only look for features on which we have to work.

```
final_train.columns
```

```
Index(['Year', 'Kilometers_Driven', 'Owner_Type', 'Seats', 'Price',
       'Mileage(km/kg)', 'Engine(CC)', 'Power(bhp)', 'Location_Bangalore',
       'Location_Chennai', 'Location_Coimbatore', 'Location_Delhi',
       'Location_Hyderabad', 'Location_Jaipur', 'Location_Kochi',
       'Location_Kolkata', 'Location_Mumbai', 'Location_Pune',
       'Fuel_Type_Diesel', 'Fuel_Type_LPG', 'Fuel_Type_Petrol',
       'Transmission_Manual'],
      dtype='object')
```

```
X = final_train.loc[:,['Year', 'Kilometers_Driven', 'Owner_Type', 'Seats',
       'Mileage(km/kg)', 'Engine(CC)', 'Power(bhp)',
       'Location_Bangalore', 'Location_Chennai', 'Location_Coimbatore',
       'Location_Delhi', 'Location_Hyderabad', 'Location_Jaipur',
       'Location_Kochi', 'Location_Kolkata', 'Location_Mumbai',
       'Location_Pune', 'Fuel_Type_Diesel', 'Fuel_Type_LPG',
       'Fuel_Type_Petrol', 'Transmission_Manual']]
X.shape
```

```
(5872, 21)
```

```
y = final_train.loc[:,['Power(bhp)']]
y.head()
```

	Power(bhp)
0	58.16
1	126.20
2	88.70
3	88.76
4	140.80

```
from sklearn.ensemble import ExtraTreesRegressor
selection= ExtraTreesRegressor()
selection.fit(X,y)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning
  This is separate from the ipykernel package so we can avoid doing imports until
ExtraTreesRegressor()
```

Build it (Model)

First we are spliting the data to train and test for the model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

Now lets try the Random Forest Regressor

```
#Now we will be using Random Forest Regressor (for better accuracy)

from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor()
rf_reg.fit(X_train, y_train)
y_pred= rf_reg.predict(X_test)
print("Accuracy on Traing set: ",rf_reg.score(X_train,y_train))
print("Accuracy on Testing set: ",rf_reg.score(X_test,y_test))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DataConversionWarning
  """
Accuracy on Traing set:  0.9999346196941928
Accuracy on Testing set:  0.9999243664535357
```

```
X_test.head()
rf_reg.predict([[2016,47000,1,7.0,12.80,2494.0,102.00,0,0,0,0,0,0,0,0,0,1,1,0,0,1]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
  "X does not have valid feature names, but"
array([102.])
```

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.