Used Car Price Predictions

Prepared by:

Purva Sonsare

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my SME (Subject Matter Expert) Mohd Kashif as well as Flip Robo Technologies who gave me the opportunity to do this project on Surprise Housing Price Prediction, which also helped me in doing lots of research wherein I came to know about so many new things. Also, I have utilized a few external resources that helped me to complete the project. I ensured that I learn from the samples and modify things according to my project requirement. All the external resources that were used in creating this project are listed below:

1) https://www.google.com/
2) https://www.youtube.com/
3) https://scikit-learn.org/stable/user_guide.html
4) https://github.com/
5) https://www.kaggle.com/
6) https://medium.com/
7) https://towardsdatascience.com/
8) https://www.analyticsvidhya.com/

INTRODUCTION

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.

Business Problem Framing

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 must make car price valuation model. This project contains two phases

Data Collection Phase: You must scrape at least 5000 used cars data. You can scrape more data as well, it's up to you. more the data better the model In this section You need to scrape the data of used cars from websites (Olx, cardekho, Cars24 etc.) You need web scraping for this. You must fetch data for different locations. The number of columns for data doesn't have limit, it's up to you and your creativity. 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. Try to include all types of cars in your data for example- SUV, Sedans, Coupe, minivan, Hatchback.

Model Building Phase: After collecting the data, you need to build-processing steps. Try different models with different hyper parameters and select the best model

Conceptual Background of the Domain Problem

There are lots of individuals who are interested in the used car market at some points in their life because they wanted to sell their car or buy a used car. In this process, it's a big corner to pay too much or sell less than its market value.
There are one of the biggest target groups that can be interested in results of this study. If used car sellers better understand what makes a car desirable, what are the important features for a used car, then they may consider this knowledge and offer a better service.

Data Sources and their formats

The data is been Scraped from carsdekho website using beautiful soup and selenium web driver , it was enable to scrape 12000 cars data Screen shot of code is shared below

Code:

```
url = "https://www.cardekho.com/"
driver.get(url)
time.sleep(3)
```

```
#link for city
used_cars = driver.find_element_by_xpath("/html/body/div[2]/div/header/div[2]/div/div/nav/ul/li[2]/ul/li[2]/a")
```

```python
Car_brand = []
car_price = []
kms_driven = []
fuel = []
transmission = []
city_url = []
car_urls = []
for i in city:
    driver.get(i)
    time.sleep(3)

    for _ in range(150):
        time.sleep(1)
        driver.execute_script("window.scrollBy(0,1000)")
    current_url = driver.current_url
    car = driver.find_elements_by_xpath("//div[@class='gsc_col-xs-7  carsName']/a")
    for i in car:
        car_urls.append(i.get_attribute('href'))
    #fetching
    for i in driver.find_elements_by_xpath("//div[@class='gsc_col-xs-7  carsName']/a"):
        Car_brand.append(i.text)
        city_url.append(current_url)
    #
    for i in driver.find_elements_by_xpath("//span[@class='amnt ']"):
        car_price.append(i.text)
    #
    for i in driver.find_elements_by_xpath("//div[@class='truncate dotlist']//span[1]"):
        kms_driven.append(i.text)
    #
    for i in driver.find_elements_by_xpath("//div[@class='truncate dotlist']//span[2]"):
        fuel.append(i.text)
    #
    for i in driver.find_elements_by_xpath("//div[@class='truncate dotlist']//span[3]"):
        transmission.append(i.text)
```

Output:

```python
#creating a dataframe
data = list(zip(Car_brand,fuel,kms_driven,engine_disp,transmission,milage,seats,color,max_power,front_brake,rear_brake,
                cargo_vol,height,width,length,weight,insp_score,top_speed,city_url,car_price))
df = pd.DataFrame(data, columns = ["Car_Name","Fuel_type","Running_in_kms","Endine_disp", "Gear_transmission",
                                   "Milage_in_km/ltr","Seating_cap","color","Max_power","front_brake_type","rear_brake_type",
                                   "cargo_volume","height","width","length","Weight","Insp_score","top_speed","City_url",
                                   "Car_price"])
df
```

	Car_Name	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_
0	2014 Maruti Wagon R	Petrol	1,31,125 kms	998	Manual	21.79 kmpl	5	Grey	67.05	Disc	
1	2015 Hyundai Grand i10	Petrol	73,875 kms	1197	Manual	18.9 kmpl	5	White	82	Disc	
2	2018 Volkswagen Vento	Diesel	97,922 kms	1498	Manual	22.27 kmpl	5	White	108.6	Ventilated Disc	
3	2019 Maruti S-Presso	Petrol	24,230 kms	998	Manual	21.7 kmpl	5	Red	67.05	Ventilated Disc	
4	2017 Maruti Wagon R	Petrol	41,174 kms	998	Automatic	20.51 kmpl	5	Grey	67	Ventilated Disc	
...	...	...	...	...	...	...	...	...	...	...	
12603	2014 Land Rover Freelander 2	Diesel	80,000 kms	2179	Automatic	12.39 kmpl	5	White	147.51	Disc	
12604	2011 Audi RS5	Petrol	55,000 kms	2894	Automatic	11.05 kmpl	4	Red	444	Ventilated Disc	Ventilatec
12605	2020 Hyundai Grand i10 Nios	CNG	40,000 kms	1197	Manual	0.0 km/kg	5	-	68.05	Disc	
12606	2021 Hyundai Xcent	CNG	15,000 kms	1197	Manual	0.0 km/kg	5	-	65.39	Disc	
12607	2011 Skoda Laura	Petrol	50,000 kms	1798	Manual	13.4 kmpl	5	-	157.8	Disc	

12608 rows × 20 columns

Exploratory Data Analysis

```
df.drop('Unnamed: 0', axis=1, inplace=True)
```

```
df.shape
```

```
(12608, 20)
```

After importing and loading the dataset I have checked the shape of data and removed Unnamed Column as it was just giving index.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12608 entries, 0 to 12607
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Car_Name           12608 non-null  object
 1   Fuel_type          12608 non-null  object
 2   Running_in_kms     12608 non-null  object
 3   Endine_disp        12608 non-null  object
 4   Gear_transmission  12608 non-null  object
 5   Milage_in_km/ltr   12608 non-null  object
 6   Seating_cap        12553 non-null  object
 7   color              12608 non-null  object
 8   Max_power          12607 non-null  object
 9   front_brake_type   12532 non-null  object
 10  rear_brake_type    12532 non-null  object
 11  cargo_volume       12161 non-null  object
 12  height             12552 non-null  object
 13  width              12552 non-null  object
 14  length             12552 non-null  object
 15  Weight             12571 non-null  object
 16  Insp_score         12608 non-null  object
 17  top_speed          10810 non-null  object
 18  City_url           12608 non-null  object
 19  Car_price          12608 non-null  object
dtypes: object(20)
memory usage: 1.9+ MB
```

Using df.info(), I have checked information about a DataFrame including the index dtype and columns, non-null values and memory usage.
Shape of data is 12608 rows and 20 columns

```
df.columns     # column names of our dataset
```

```
Index(['Car_Name', 'Fuel_type', 'Running_in_kms', 'Endine_disp',
       'Gear_transmission', 'Milage_in_km/ltr', 'Seating_cap', 'color',
       'Max_power', 'front_brake_type', 'rear_brake_type', 'cargo_volume',
       'height', 'width', 'length', 'Weight', 'Insp_score', 'top_speed',
       'City_url', 'Car_price'],
      dtype='object')
```

Using df.columns checked the names of all columns

```
#Replacing unnecessary entries
df.replace('-',np.nan, inplace = True)
df.replace('null ',np.nan, inplace = True)
```

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

```
Car_Name                 0
City_url                 0
Gear_transmission        0
Car_price                0
Running_in_kms           0
Fuel_type                0
Milage_in_km/ltr        29
Endine_disp             60
Seating_cap            104
Max_power              145
front_brake_type       214
rear_brake_type        215
height                 254
length                 254
width                  255
color                  274
top_speed             4316
Weight                6074
cargo_volume          8388
Insp_score           10876
dtype: int64
```

```
df.drop(['cargo_volume','Insp_score'], axis=1, inplace=True)
```

Checking null values of dataset by isnull() method
I have replaced blank and hyphen columns with nan values to get the total and proper count of missing values

Also removed the columns where missing values were greater than half of its total count
Then shape of the data was changed to 12608 rows and 18 columns after removing 'cargo_volume' and 'Insp_score'

Car_Name:

Since Car_name column has year of manufacture, car model and car name all together so we have to separate them.

```
#Extracting manufacturing year and car name from Car_Name
df['Manu_year'] = df['Car_Name'].str[0:4]
df['car_name'] = df['Car_Name'].str[4:]
df.drop(columns = 'Car_Name', inplace = True)
```

```
df['Car_Brand'] = df.car_name.str.split(' ').str.get(1)
df['Car_Model'] = df.car_name.str.split(' ').str[2:]
df['Car_Model'] = df['Car_Model'].apply(lambda x: ','.join(map(str, x)))
df['Car_Model'] = df['Car_Model'].str.replace(',',' ')
df.drop(columns = 'car_name', inplace = True)
```

Car_Price:

Since Car_Price is our target it should be in the format of continuous data, so w

```
# replacing the unit with numerical values and removing the ','
df['car_price'] = df['Car_price'].str.replace('Lakh','100000')
df['car_price'] = df['car_price'].str.replace(',','')
df['car_price'] = df['car_price'].str.replace('Cr','10000000')
```

```
# splitting the unit(numerical) ans amount of car price
df[['a','b']] = df.car_price.str.split(expand=True)
```

```
# changing the datatype to float
df['a'] = df['a'].astype('float')
df['b'] = df['b'].astype('float')
```

```
df['b']=df['b'].fillna(value = 1)
```

```
df['car_price'] = df['a'] * df['b']
df.drop(columns = ['Car_price','a','b'], inplace = True)
```

Also performed feature extraction on some columns.
Screen shots attached for reference
From car_name I have extracted the manufacturing year, brand name and car model.
From Car_price first I replaced the units with numerical values
Then separated the unit and amount value and checked for missing values in unit column and replaced those values with 1
Then multiplied both separated values and saved

Milage_in_km/ltr:

In Milage_in_km/ltr column the data type is object so we have to change this to float type.

```
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('kmpl','')
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('km/kg','')
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('km/hr','')

df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].astype('float')
```

Engine_disp:

The column 'Endine_disp' should be continuous column so i will convert it to float datatype.

```
df.Endine_disp = df.Endine_disp.astype('float')
```

Converting the data type of columns height, width and length to float datatype:

```
df['height'] = df['height'].str.replace(',','')
df['height'] = df['height'].str[0:4]
df['width'] = df['width'].str.replace(',','')
df['length'] = df['length'].str.replace(',','')
df.height = df.height.astype('float')
df.width = df.width.astype('float')
df.length = df.length.astype('float')
```

top_speed:

Let's change the datatype of top_speed column to float.

```
#getting numerical values from top_speed column
df['top_speed'] = df['top_speed'].str[0:3]
df.top_speed = df.top_speed.astype('float')
```

Similarly, I have extracted some meaningful data from columns like Milage_in_kms/ltr, Engine_disp, top_speed
Also changed their datatype from object to float.
Screen shots are attached for reference.
Extracted city name from urls
Separated units and values in columns like milage_in_kms/ltr, top_speed, etc
Also replaced similar values under one head in all columns.
Screen shot of all the above steps are attached in next two slides.

Weight:

The datatype of Weight has to be changed to float.

```
#getting numerical values from Weight column
df['Weight'] = df['Weight'].str.replace(',','')
df['Weight'] = df['Weight'].str[0:4]

df.Weight = df.Weight.astype('float')
```

City_url:

Let's extract city name from city url column.

```
#Checking value counts of City_url column
df.City_url.value_counts()
```

```
https://www.cardekho.com/used-cars+in+delhi-ncr    1490
https://www.cardekho.com/used-cars+in+bangalore    1486
https://www.cardekho.com/used-cars+in+mumbai       1478
https://www.cardekho.com/used-cars+in+new-delhi    1473
https://www.cardekho.com/used-cars+in+pune         1239
https://www.cardekho.com/used-cars+in+gurgaon      1040
https://www.cardekho.com/used-cars+in+noida         982
https://www.cardekho.com/used-cars+in+hyderabad     918
https://www.cardekho.com/used-cars+in+chennai       836
https://www.cardekho.com/used-cars+in+kolkata       595
https://www.cardekho.com/used-cars+in+ahmedabad     579
https://www.cardekho.com/used-cars+in+jaipur        492
Name: City_url, dtype: int64
```

```
#Replacing city names from city urls
df['city_name'] = df.City_url.replace('https://www.cardekho.com/used-cars+in+bangalore', 'Bangalore')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+mumbai', 'mumbai')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+chennai', 'Chennai')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+hyderabad', 'hyderabad')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+pune', 'pune')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+delhi-ncr', 'delhi-ncr')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+ahmedabad', 'ahmedabad')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+gurgaon', 'gurgaon')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+noida', 'noida')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+kolkata', 'kolkata')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+jaipur', 'jaipur')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+new-delhi', 'new-delhi')
```

```
#Let's check the value count again
df['city_name'].value_counts()
```

```
delhi-ncr    1490
Bangalore    1486
mumbai       1478
new-delhi    1473
pune         1239
gurgaon      1040
noida         982
hyderabad     918
Chennai       836
kolkata       595
ahmedabad     579
jaipur        492
Name: city_name, dtype: int64
```

Manu_Year:

Let's extract car age from manufactured year.

```
df.Manu_year = df.Manu_year.astype('float')
df['Car_age'] = 2022 - df['Manu_year']
df.drop(columns = 'Manu_year', inplace = True)
```

Max_power:

We have to change the datatype of Max_power column to float datatype.

```
#getting numerical values from column Max_power and converting them to float type
df['Max_power'] = df['Max_power'].str[0:5]
```

```
df['Max_power'] = df['Max_power'].str[0:5]
df['Max_power'] = df['Max_power'].str.replace('PS','')
df['Max_power'] = df['Max_power'].str.replace('ps','')
df['Max_power'] = df['Max_power'].str.replace('Bh','')
df['Max_power'] = df['Max_power'].str.replace('P','')
```

```
df.Max_power = df.Max_power.astype('float')
```

```
df["front_brake_type"].replace("Solid Disc","Disc",inplace=True)
df["front_brake_type"].replace("Disk","Disc",inplace=True)
df["front_brake_type"].replace("Discs","Disc",inplace=True)
df["front_brake_type"].replace("Disc, 236 mm","Disc",inplace=True)
df["front_brake_type"].replace("disc","Disc",inplace=True)

df["front_brake_type"].replace("Ventilated Discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated DIsc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated Disk","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("264mm Ventilated discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Vantilated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Disc,internally ventilated","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventlated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventillated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventillated Discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Booster assisted ventilated disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated disc","Ventilated Disc",inplace=True)
```

front_brake_type:

Let's group the similar entries in this column.

```
#Checking the value counts of front_brake_type
df['front_brake_type'].value_counts()
```

```
Disc                                        6902
Ventilated Disc                             4785
Solid Disc                                   181
Ventilated Discs                             141
Disc & Caliper Type                           83
Disk                                          73
Ventilated DIsc                               51
Ventilated discs                              33
Drum                                          25
Ventilated Disk                               17
Multilateral Disc                             14
264mm Ventilated discs                        13
Electric Parking Brake                        11
Vantilated Disc                               10
Vacuum assisted hydraulic dual circuit w       7
Disc & Drum                                    7
Discs                                          6
Disc,internally ventilated                     6
Disc, 236 mm                                   5
disc                                           4
Ventillated Disc                               4
Ventlated Disc                                 4
Ventillated Discs                              3
Carbon ceramic                                 2
Booster assisted ventilated disc               2
Mechanical-hydraulic dual circuit              1
Ventilated disc                                1
```

Handling Missing Values

```
#Replacing nan values by median, mode, mean as given below
for col in ['Endine_disp','Milage_in_km/ltr','Max_power','height','width','Weight','top_speed']:
    df[col] = df[col].fillna(df[col].median())
for col in ['length']:
    df[col] = df[col].fillna(df[col].mean())
for col1 in ['Seating_cap','color','front_brake_type','rear_brake_type']:
    df[col1] = df[col1].fillna(df[col1].mode()[0])
```

Checking Unique Values and Value Counts

After replacing, cleaning and combining similar values under one head I have got the unique values and value counts
Screen shots are attached

```
# checking Value counts
for col in df:
    print(col)
    print(df[col].value_counts())
    print('\n')
```

```
Fuel_type
Petrol      7056
Diesel      5422
CNG           92
LPG           27
Electric      11
Name: Fuel_type, dtype: int64
```

```
df.nunique().sort_values().to_frame()
```

	0
Gear_transmission	2
Fuel_type	5
Seating_cap	8
city_name	12
front_brake_type	13
rear_brake_type	17
Car_age	25
Car_Brand	35
top_speed	120
Endine_disp	143
color	195
width	224
Weight	230
height	241
Car_Model	275
length	322
Max_power	376
Milage_in_km/ltr	478
car_price	1227
Running_in_kms	4577

Checking Duplicate Values and Statistical Analysis

```
df.duplicated().sum()
```

```
796
```

```
df.drop_duplicates(inplace=True)
df.shape
```

```
(11812, 20)
```

I have found 796 duplicate values and drop the them. And shape of data is changed to 11812 rows and 20 columns

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Running_in_kms	11812.0	57875.737809	3.964796e+04	200.0	33073.25	55000.00	76000.00	1080000.0
Endine_disp	11812.0	1430.912123	4.925664e+02	0.0	1197.00	1248.00	1498.00	5998.0
Milage_in_km/ltr	11812.0	19.608627	4.211795e+00	0.0	17.05	19.67	22.15	36.0
Seating_cap	11812.0	5.216136	6.919597e-01	2.0	5.00	5.00	5.00	10.0
Max_power	11812.0	99.522913	4.379747e+01	32.5	74.00	86.80	110.00	641.0
height	11812.0	1563.584067	1.105008e+02	148.0	1489.00	1520.00	1630.00	1995.0
width	11812.0	1717.467237	1.250578e+02	1410.0	1665.00	1700.00	1765.00	2220.0
length	11812.0	4078.903497	3.971362e+02	3099.0	3765.00	3995.00	4395.00	5295.0
Weight	11812.0	1599.351507	3.182351e+02	1000.0	1505.00	1520.00	1550.00	3350.0
top_speed	11812.0	171.483237	2.124721e+01	86.0	163.00	169.00	173.00	327.0
car_price	11812.0	756326.558838	1.004360e+06	30000.0	350000.00	510000.00	740000.00	30800000.0
Car_age	11812.0	7.111158	3.138292e+00	1.0	5.00	7.00	9.00	36.0

- I have used describe method for statistical analysis and found following observations
- All columns has same count ie. 11812.
- columns are: Running_in_kms, Endine_disp, Seating_cap, Max_power, height,width,Milage_in_km/ltr length, Weight, top_speed,car_price,Car_age. All these columns are right skewed.
- Larger difference between 75% and Max value indicates presence of outliers.

Visualizations

Univariate Analysis

```
# checking for categorical columns
categorical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_columns.append(i)
print("Categorical Col:",categorical_columns)

# checking for numerical columns
numerical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
print("Numerical Col:",numerical_columns)
```

```
Categorical Col: ['Fuel_type', 'Gear_transmission', 'color', 'front_brake_type', 'rear_brake_type', 'Car_Brand', 'Car_Model',
'city_name']
Numerical Col: ['Running_in_kms', 'Endine_disp', 'Milage_in_km/ltr', 'Seating_cap', 'Max_power', 'height', 'width', 'length',
'Weight', 'top_speed', 'car_price', 'Car_age']
```

Univariate analysis of categorical columns was done after separating the columns. Screenshot of code is attached.

- Maximum cars are petrol driven and also diesel driven.
- Maximum cars are with Manual gear transmission.
- Disc front brake cars are more in number followed by Ventilated Disc.
- Drum rare break cars are more in number.
- Maximum cars under sale are Maruti followed by Hyundai.
- In Bangalore, Delhi-NCR, Mumbai and New Delhi we can find maximum cars for sale. Since these are most populated places.

Bivariate Analysis

```
col=['Running_in_kms', 'Endine_disp', 'Milage_in_km/ltr',
     'Max_power', 'height', 'width', 'length', 'Weight', 'top_speed']
```

```
#regplot for numerical columns
plt.figure(figsize=(20,40))
for i in range(len(col)):
    plt.subplot(6,2,i+1)
    sns.regplot(x=df[col[i]] , y=df['car_price'],color="g")
    plt.title(f"car_price VS {col[i]}",fontsize=20)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col[i],fontsize = 20)
    plt.ylabel('car_price',fontsize = 20)
    plt.tight_layout()
```

Observation:

- Maximum cars are having below 20k driven kms. And car price is high for less driven cars.
- Maximum cars are having 1000-3000 Endine_disp. And car price is high for 3000 Endine_disp. Maximum cars are having milage of 10-25kms. And ,milage has no proper relation with car price.
- As Max_power is increasing car price is also increasing.
- Car_price has no proper relation with height.
- As the width is increasing car price is also increasing.
- As length is increasing car price is also increasing.

- Weight also has linear relationship with car price.
- As top_speed is increasing car price is also increasing.

Comparing car price and columns in list col1

```
col1=['Seating_cap','Car_age']
```

```
#stripplot for numerical columns
plt.figure(figsize=(20,5))
for i in range(len(col1)):
    plt.subplot(1,2,i+1)
    sns.stripplot(x=df[col1[i]] , y=df['car_price'])
    plt.title(f"car_price VS {col1[i]}",fontsize=20)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col1[i],fontsize = 20)
    plt.ylabel('car_price',fontsize = 20)
    plt.tight_layout()
```

Observations

- Cars with 5 and 4 seats are having highest price.
- As the age of the car increases the car price decreases.

Comparing car price and columns in list col2

```
col2=['Fuel_type', 'Gear_transmission', 'front_brake_type', 'rear_brake_type', 'Car_Brand', 'city_name']
```

```
#Bar plot for all categorical columns
plt.figure(figsize=(20,30))
for i in range(len(col2)):
    plt.subplot(3,2,i+1)
    sns.barplot(y=df['car_price'],x=df[col2[i]])
    plt.title(f"car_price VS {col2[i]}",fontsize=25)
    plt.xticks(rotation=90,fontsize=15)
    plt.yticks(rotation=0,fontsize=15)
    plt.xlabel(col2[i],fontsize = 20)
    plt.ylabel('car_price',fontsize = 20)
    plt.tight_layout()
```

Observations:

- For Diesel and Electric cars the price is high compared to Petrol,LPG and CNG.
- Cars with automatic gear are costlier than manual gear cars.
- Cars with Carbon Ceramic front break are costlier compared to other cars.
- Lamborghini brand cars are having highset sale price.
- In Bangalore, Hyderabad and delhi-ncr the car prices are high as they are highly populated cities.

Checking Correlation

```
#Checking correlation of the dataset
df.corr()
```

	Running_in_kms	Endine_disp	Milage_in_km/ltr	Seating_cap	Max_power	height	width	length	Weight	top_speed	car_price
Running_in_kms	1.000000	0.192793	-0.058002	0.160492	0.053823	0.083693	0.092965	0.160971	0.192286	0.041468	-0.126317
Endine_disp	0.192793	1.000000	-0.576329	0.446198	0.853103	0.374953	0.713525	0.798983	0.732165	0.571256	0.581814
Milage_in_km/ltr	-0.058002	-0.576329	1.000000	-0.329186	-0.533664	-0.285201	-0.374317	-0.475659	-0.494074	-0.294253	-0.322607
Seating_cap	0.160492	0.446198	-0.329186	1.000000	0.173218	0.673171	0.233186	0.341037	0.513646	-0.111782	0.097470
Max_power	0.053823	0.853103	-0.533664	0.173218	1.000000	0.165611	0.760331	0.780429	0.616637	0.745312	0.731215
height	0.083693	0.374953	-0.285201	0.673171	0.165611	1.000000	0.228576	0.172339	0.447966	-0.151470	0.157324
width	0.092965	0.713525	-0.374317	0.233186	0.760331	0.228576	1.000000	0.780510	0.638814	0.636092	0.578914
length	0.160971	0.798983	-0.475659	0.341037	0.780429	0.172339	0.780510	1.000000	0.648334	0.645138	0.511190
Weight	0.192286	0.732165	-0.494074	0.513646	0.616637	0.447966	0.638814	0.648334	1.000000	0.406561	0.419034
top_speed	0.041468	0.571256	-0.294253	-0.111782	0.745312	-0.151470	0.636092	0.645138	0.406561	1.000000	0.517921
car_price	-0.126317	0.581814	-0.322607	0.097470	0.731215	0.157324	0.578914	0.511190	0.419034	0.517921	1.000000
Car_age	0.425396	0.099144	-0.234918	0.005584	-0.005857	-0.100875	-0.116232	0.014766	0.065655	0.037051	-0.262931

```
#Plotting heatmap for visualizing the correlation
plt.figure(figsize=(15,10))
sns.heatmap(df.corr() ,linewidth=0.5,linecolor='black',fmt='.0%',cmap='rocket',annot=True)
plt.show()
```

Observations:

1. Highly and positively correlated: Max_power, width, Endine_disp, top_speed, weight, length
2. Negatively and least correlated: Running_in_kms, Car_age

```
# plotting bar plot of correlation
plt.style.use('seaborn-white')
df_corr = df.corr()
plt.figure(figsize=(15,8))
df_corr['car_price'].sort_values(ascending=False).drop('car_price').plot.bar()
plt.title("Correlation of Features vs Label\n", fontsize=16)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=12)
plt.show()
```

we can see from above graph that max_power has highest correlation with car_price followed by, endine-disp and least correlation is with Seating_cap

Encoding the Categorical data

Using Label Encoder I have transformed all categorical data into numerical form and checked the shape of data
screen shot is attached below

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
list1=['Fuel_type', 'Gear_transmission', 'color', 'front_brake_type', 'rear_brake_type', 'Car_Brand', 'Car_Model', 'city_name']
for val in list1:
    df[val]=le.fit_transform(df[val].astype(str))
```

```
df.head()
```

	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_type	height	w
0	4	131125.0	998.0	1	21.79	5.0	70	67.05	1	7	1675.0	16
1	4	73875.0	1197.0	1	18.90	5.0	176	82.00	1	7	1520.0	16
2	1	97922.0	1498.0	1	22.27	5.0	176	108.60	12	7	1467.0	16
3	4	24230.0	998.0	1	21.70	5.0	135	67.05	12	7	1564.0	15
4	4	41174.0	998.0	0	20.51	5.0	70	67.00	12	7	1700.0	14

Hence all our categorical data is converted to numerical format

```
df.shape  # checking the shape
```

```
(11812, 20)
```

Checking Outliers

```
fig, ax = plt.subplots(ncols=4, nrows=5, figsize=(15,15))
index = 0
features = df.drop("car_price", axis=1)
ax = ax.flatten()
for col, value in features.items():
    sns.boxplot(y=col, data=df, ax=ax[index], palette='plasma')
    index += 1
plt.tight_layout(pad=0.1, w_pad=0.5, h_pad=1.0)
plt.show()
```

From above box plots we can see that columns having outliers are: Running_in_kms, Endine_disp, Milage_in_km/ltr, Seating_cap, height, width, weight, top_speed, car_age. I have not considered encoded columns for outliers

Removing Outliers

```
z = np.array(abs(zscore(df)))
threshold = 3
print(np.where(z>3))
df1_new=df[(z<3).all(axis=1)]
df1_new.head(10)
```

```
(array([   17,    29,    60, ..., 11808, 11809, 11810], dtype=int64), array([ 5,  5, 14, ..., 14,  4,  4], dtype=int64))
```

	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_type	height	w
0	4	131125.0	998.0	1	21.79	5.0	70	67.05	1	7	1675.0	16
1	4	73875.0	1197.0	1	18.90	5.0	176	82.00	1	7	1520.0	16
2	1	97922.0	1498.0	1	22.27	5.0	176	108.60	12	7	1467.0	16
3	4	24230.0	998.0	1	21.70	5.0	135	67.05	12	7	1564.0	15
4	4	41174.0	998.0	0	20.51	5.0	70	67.00	12	7	1700.0	14
5	4	57341.0	1196.0	1	16.11	5.0	176	72.42	1	7	1825.0	14
6	4	38647.0	1497.0	1	17.00	5.0	103	118.00	1	7	1480.0	16
7	4	17116.0	1196.0	1	16.11	5.0	176	72.42	1	7	1825.0	14
8	4	45347.0	1196.0	1	15.37	5.0	176	73.00	12	7	1800.0	14
9	4	45203.0	1197.0	1	20.40	5.0	176	81.80	12	7	1530.0	17

```
print("Shape of old data",df.shape)
print("Shape of new data",df1_new.shape)
```

```
Shape of old data (11812, 20)
Shape of new data (10611, 20)
```

```
dataloss= ((11812-10611)/10611)*100
print("PErcentage data loss is", dataloss)
```

```
PErcentage data loss is 11.3184431250589
```

After removing outliers we are having 11% data loss which is more than enough because we still have a large amount of data

```
df = df1_new.copy()        # copying the updated data to old dataframe
```

- I have removed outliers using zscore method.
- Treshhold value is 3
- Data loss in this process is 11%
- But as the data is large we can afford this loss
- New shape of data is 10611 rows and 20 rows

Checking Skewness

```
fig, ax = plt.subplots(ncols=3, nrows=4, figsize=(10,10))
index = 0
ax = ax.flatten()
for col, value in df[numerical_columns].items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade": True})
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.4, h_pad=1.0)
plt.show()
```

Observations

We see from above plots that skewness is present in some columns which are:height, Endine_disp, top_speed, Max_power, Weight, seating_cap again I am ignoring the encoded coluumns as there is no point in treating their skewness

Splitting the dataframe in Feature and target Variable

```
x = df.drop('car_price', axis=1)
y = df['car_price']
```

I have splitted the data in target variable and feature variable. X denoting feature varible and Y denoting the target variable

Standardization of Data

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x.head(10)
```

	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_type	hei
0	0.813936	2.533725	-1.001163	0.488901	0.445365	-0.276059	-0.946033	-0.869946	-0.796984	0.302389	1.3517
1	0.813936	0.617880	-0.408728	0.488901	-0.374652	-0.276059	1.069231	-0.345220	-0.796984	0.302389	-0.3542
2	-1.219659	1.422602	0.487366	0.488901	0.581561	-0.276059	1.069231	0.588406	1.259675	0.302389	-0.9376
3	0.813936	-1.043468	-1.001163	0.488901	0.419828	-0.276059	0.289742	-0.869946	1.259675	0.302389	0.1300
4	0.813936	-0.476444	-1.001163	-2.045405	0.082174	-0.276059	-0.946033	-0.871701	1.259675	0.302389	1.6268
5	0.813936	0.064577	-0.411705	0.488901	-1.166294	-0.276059	1.069231	-0.681466	-0.796984	0.302389	3.0026
6	0.813936	-0.561009	0.484389	0.488901	-0.913763	-0.276059	-0.318639	0.918334	-0.796984	0.302389	-0.7945
7	0.813936	-1.281535	-0.411705	0.488901	-1.166294	-0.276059	1.069231	-0.681466	-0.796984	0.302389	3.0026
8	0.813936	-0.336797	-0.411705	0.488901	-1.376264	-0.276059	1.069231	-0.661108	1.259675	0.302389	2.7275
9	0.813936	-0.341616	-0.408728	0.488901	0.050962	-0.276059	1.069231	-0.352240	1.259675	0.302389	-0.2442

Using standard scaler I have scaled the feature variables. This is called Standardization of data

Removing Skewness

```
from sklearn.preprocessing import PowerTransformer
power_t = PowerTransformer(method = 'yeo-johnson')
x = pd.DataFrame(power_t.fit_transform(x), columns=x.columns)
x.head()
```

	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_type	hei
0	0.815594	2.172797	-1.231050	0.488901	0.463089	-0.179501	-1.011818	-1.041799	-0.798998	0.233973	1.3475
1	0.815594	0.689500	-0.285973	0.488901	-0.356122	-0.179501	1.203150	-0.172183	-0.798998	0.233973	-0.1340
2	-1.224995	1.355164	0.701983	0.488901	0.595777	-0.179501	1.203150	0.825968	1.258429	0.233973	-1.2354
3	0.815594	-1.095017	-1.231050	0.488901	0.438122	-0.179501	0.117998	-1.041799	1.258429	0.233973	0.4894
4	0.815594	-0.410637	-1.231050	-2.045405	0.105000	-0.179501	-1.011818	-1.045081	1.258429	0.233973	1.4782

```
x.skew().sort_values()
```

```
Seating_cap          -4.679264
Gear_transmission    -1.556725
Fuel_type            -0.414105
Weight               -0.246954
color                -0.155565
Endine_disp          -0.089712
top_speed            -0.086013
Max_power            -0.075552
city_name            -0.029820
Milage_in_km/ltr     -0.008415
Car_Model            -0.003607
length                0.016501
Car_age               0.032809
height                0.035558
Running_in_kms        0.053260
Car_Brand             0.067752
width                 0.105803
front_brake_type      0.460764
rear_brake_type       1.614980
dtype: float64
```

Observations
Hence here we have tried removing skewness using power transformation.
Ignoring the skewness of encoded variables

Principle Component Analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components=19, svd_solver="full")
x = pd.DataFrame(pca.fit_transform(x), columns=x.columns)
x.head()
```

	Fuel_type	Running_in_kms	Endine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_type	rear_brake_type	hei
0	2.061756	-1.342393	0.190407	-0.596725	-0.236805	0.455472	-0.170709	-2.070247	-0.242220	-0.886420	0.469
1	0.758319	0.197878	0.676694	-0.743719	-0.573034	1.239503	-1.403332	0.301906	0.372441	0.128419	-0.097
2	-1.551801	-1.540088	0.261161	1.460147	-0.257758	-0.995182	-0.525687	-0.710993	1.725495	1.883352	0.096
3	3.504525	0.772178	-1.161564	0.022676	0.505466	0.264872	0.816234	-0.483532	1.696550	0.318579	-0.705
4	2.750028	0.374927	-1.345852	-1.150314	0.952191	-0.205032	1.312355	-1.351385	1.740375	-1.435828	1.463

Checking Multicollinearity

```
import statsmodels.api as sm
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
def calc_vif(x):
    vif=pd.DataFrame()
    vif['variables']=x.columns
    vif["VIF FACTOR"]= [variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    return(vif)

calc_vif(x)
```

	variables	VIF FACTOR
0	Fuel_type	1.0
1	Running_in_kms	1.0
2	Endine_disp	1.0
3	Gear_transmission	1.0
4	Milage_in_km/ltr	1.0
5	Seating_cap	1.0
6	color	1.0
7	Max_power	1.0
8	front_brake_type	1.0
9	rear_brake_type	1.0
10	height	1.0
11	width	1.0
12	length	1.0
13	Weight	1.0
14	top_speed	1.0
15	Car_Brand	1.0
16	Car_Model	1.0
17	city_name	1.0
18	Car_age	1.0

As we can see all columns VIF factor is less than 10 we can proceed with this data

Model/s Development and Evaluation

```
lg  = LinearRegression()
svr = SVR(kernel='rbf')
dtr = DecisionTreeRegressor()
ridr= Ridge(alpha=1e-2, normalize=True)
las = Lasso(alpha=1e-2, normalize=True, max_iter=100)
rfr = RandomForestRegressor()
gbr = GradientBoostingRegressor(loss='quantile', n_estimators=200, max_depth=5)
abr = AdaBoostRegressor(n_estimators=300, learning_rate=1.05, random_state=42)
etr = ExtraTreesRegressor(n_estimators=200, max_features='sqrt', n_jobs=6)


model=[lg,svr,dtr,ridr,las,rfr,gbr,abr,etr]

for m in model:
    print('\n')
    print('******************',m,'*******************')
    print('\n')
    m.fit(x_train,y_train)
    m.score(x_train,y_train)

    pred_train=m.predict(x_train)
    pred_test=m.predict(x_test)

    # calculating scores
    train_score=r2_score(y_train,pred_train)
    test_score=r2_score(y_test,pred_test)
    r2 = r2_score(y_test, pred_test)*100

    #Calculating cross validation score

    for i in range(2,8):
        cvs=cross_val_score(m,x,y,cv=i)

    a = max(cvs)
    cv_score=(a.mean())*100    # taking max value of cvs and calculating mean of it
# taking max value because I am assuming that r2 score of all models to be approx 100 %
```

```
    # final result comparing r2 and cross validation
    result = r2 - cv_score

    print('The training accuracy of is', train_score)
    print('The test accuracy of is', test_score)
    print("R2 Score is:", r2)
    print("Cross Validation Scores",cvs)
    print("Maximum of cvs:" , a)
    print("Cross Validation Score:", cv_score)
    print("R2 Score - Cross Validation Score is", result)
    print("Error Calculations:")
    print("Mean absolute error :",mean_absolute_error(y_test,pred_test))
    print("Mean squared error : ",mean_squared_error(y_test, pred_test))
    print("Root mean squared error:", np.sqrt(mean_squared_error(y_test,pred_test)))
```

```
****************** LinearRegression() *******************


The training accuracy of is 0.6793802755163227
The test accuracy of is 0.7266077551703043
R2 Score is: 72.66077551703043
Cross Validation Scores [0.67986735 0.63391374 0.71203933 0.66042975 0.67141341 0.69250801
 0.63837387]
Maximum of cvs: 0.7120393268944584
Cross Validation Score: 71.20393268944584
R2 Score - Cross Validation Score is 1.4568428275845946
Error Calculations:
Mean absolute error : 153455.99981921114
Mean squared error :  53880208248.895546
Root mean squared error: 232121.1068578115
```

```
Performance_table=pd.DataFrame({'Model':['LR','SVR','DTR','ridge','lasso','RFR','GBR','ABR','ETR'],
            'R2 Score': [72.66,-7.47,82.44,72.65,72.66,93.16,82.58,38.35,92.43],
            'CV Score': [71.20,-2.07,90.64,71.08,71.20,95.65,88.17,66.16,95.11],
            'Result' :[1.45,-5.39,-8.19,1.57,1.5,-2.48,-5.58,-27.80,-2.67],
           'RMSE':[232121.1,460237.75,185993,232143,232121.1,116067.1,185243.2,348553.7,122108.7]})
Performance_table
```

	Model	R2 Score	CV Score	Result	RMSE
0	LR	72.66	71.20	1.45	232121.10
1	SVR	-7.47	-2.07	-5.39	460237.75
2	DTR	82.44	90.64	-8.19	185993.00
3	ridge	72.65	71.08	1.57	232143.00
4	lasso	72.66	71.20	1.50	232121.10
5	RFR	93.16	95.65	-2.48	116067.10
6	GBR	82.58	88.17	-5.58	185243.20
7	ABR	38.35	66.16	-27.80	348553.70
8	ETR	92.43	95.11	-2.67	122108.70

Observations

- All results are stored in a table format for easy view.
- From that I found that Random Forest Regressor gives the best results

Hypertuning

```
# chossing RandomForestRegressor()
parameters={'max_features':['auto','sqrt','log2'],'criterion':['mse','mae'],'max_depth':np.arange(2,10)}
```

```
GCV=GridSearchCV(RandomForestRegressor(),parameters,cv=3)
```

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

```
GridSearchCV(cv=3, estimator=RandomForestRegressor(),
             param_grid={'criterion': ['mse', 'mae'],
                         'max_depth': array([2, 3, 4, 5, 6, 7, 8, 9]),
                         'max_features': ['auto', 'sqrt', 'log2']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
GCV.best_params_
```

```
{'criterion': 'mse', 'max_depth': 9, 'max_features': 'auto'}
```

```
GCV.best_score_
```

```
0.8946933108141853
```

```
final_model=RandomForestRegressor(criterion='mse', max_depth= 9, max_features= 'auto')
final_model.fit(x_train,y_train)
pred_final=final_model.predict(x_test)
rfs=r2_score(y_test,pred_final)
print('R2 Score', rfs*100)
```

```
R2 Score 91.6745357954153
```

Using Grid Search CV I have used hypertuning to increase the accuracy
Of selected model
And chose the hypertuned parameters are given in screen shot and got the final R2 score as 92%

Saving the Model

```
filename = "CarPricePredictons.pkl"
joblib.dump(final_model, filename)
```

```
['CarPricePredictons.pkl']
```

Plotting the Results

```
# Loading the saved model
model=joblib.load("CarPricePredictons.pkl")

#Prediction
prediction = model.predict(x_test)
prediction
```

```
array([1100503.21644042,  304772.97139179,  617119.41242517, ...,
       1323444.86612189,  437211.49361068,  895658.96139766])
```

```
pd.DataFrame([model.predict(x_test)[:],y_test[:]],index=["Predicted","Actual"])
```

	0	1	2	3	4	5	6	7	8	9
Predicted	1.100503e+06	304772.971392	617119.412425	2.205989e+06	1.026858e+06	566031.172878	394643.056552	400218.344586	1.741956e+06	1.315188e+06
Actual	7.670000e+05	300000.000000	539000.000000	1.700000e+06	1.295000e+06	310000.000000	422000.000000	289000.000000	1.699000e+06	1.325000e+06

Saved the model using joblib
After saving I have loaded the saved model to compare the actual and predicted values of target variable that is car price
Also plotted the same as shown

```
plt.figure(figsize=(10,5))
plt.scatter(y_test, prediction, c='crimson')
p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
plt.title("RandomForestRegressor")
plt.show()
```

Conclusion

- In this project report, we have used machine learning algorithms to predict the used car price. We have mentioned the step by step procedure to analyse the dataset and finding the correlation between the features.
- Thus we can select the features which are correlated to each other and are independent in nature. The power of visualization has helped us in understanding the data by graphical representation it has made me to understand what data is trying to say.
- Data cleaning is one of the most important steps to remove unrealistic values and unnecessary values.
- These feature set were then given as an input to five algorithms and a hyper parameter tunning was done to the best model and the accuracy has been improved. Hence we calculated the performance of each model using different

performance metrics and compared them based on these metrics.

- Then we have also saved the best model and predicted the car price. It was good that the predicted and actual values were almost same.
- To conclude, the application of machine learning in car price prediction is still at an early stage. We hope this study has moved a small step ahead in providing some methodological and empirical contributions to online platforms, and presenting an alternative approach to the valuation of used car price.
- Future direction of research may consider incorporating additional used car data from a larger economical background with more features.

Limitations of this work and Scope for FutureWork

- First drawback is scrapping the data as it is fluctuating process.

- Followed by a greater number of outliers and skewness these two will reduce our model accuracy.
- Also, we have tried best to deal with outliers, skewness, and null values. So, it looks quite good
- that we have achieved a accuracy of 92.29% even after dealing all these drawbacks.

- Also, this study will not cover all Regression algorithms instead, it is focused on the chosen algorithm, starting from the basic ensembling techniques to the advanced ones.