Used Car Price Prediction

Submitted by:

Ali Asgar

ACKNOWLEDGMENT

- https://www.mordorintelligence.com/industry-reports/india-used-car-market
- https://www.autocarindia.com/car-news/used-car-market-to-be-twice-as-big-as-new-car-segment-by-fy2025-421140
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html
- https://www.python.org/doc/essays/blurb/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html
- https://machinelearningmastery.com/logistic-regression-for-machine-learning/
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-learn.org/stable/modules/tree.html
- https://scikit-learn.org/stable/modules/neighbors.html
- https://scikit-learn.org/stable/modules/ensemble.html#forest
- https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

INTRODUCTION

Business Problem Framing

Our client works with traders who deal in used cars. The used car market is one of the thriving industries which requires precision and understanding of the domain and also a more effective and precise prediction of the car prices to make higher profits.

Our client used to have a machine learning model which they used to predict the car price until now. Covid19 had made a significant impact on almost every being and every sector. Our client business didn't remain untouched with this pandemic. After Covid19's fear and a series of lockdowns, scenarios have changed in the used car market. Prices have changed significantly. There is more demand for individual mobility as compared to before.

Major challenges which used car market faces are no-profit/loss or market downfall. Businesses are keen to have better insights and predictions of the market and pricing to get in a healthy profit-earning situation. No-profit/loss certainly hampers the business and its working model. If the company continues to be in this state, it certainly requires more investments to be put in to overcome the situation and to keep the business running.

With machine learning, we are trying to construct a model which can help to predict the price of a car. These predicted prices will help the company to have a better idea of the financial situation.

Conceptual Background of the Domain Problem

Pricing in Used Car Business!

Pricing is the key in the Used Car Business. Pricing, whether it be purchase or selling, plays a crucial role in earning high profits. With a proper prediction of prices, the company can earn a suitable profit. This situation of thriving profit put the company in a better position and also inspire it to provide its customers with various attractive and lucrative offers and facilities. This will result in better publicity and thus an enhancement in the business.

The organized sales channel witnessed significant growth in the last three years. This growth is driven by increased sales of used cars in metro cities and a rise in online sales platforms, such as CarDekho, Cars24, and Droom.
Poor price assessment, poor risk assessment, location selection, less efficient and less experienced professionals, customer dissatisfaction, etc can be the factors that can put the company in a no-profit or loss situation. The majority of the OEMs have already entered the used car market, and those who have not entered the used car market in the nascent stages have also entered the market during the last five years

Review of Literature

What is Used Car Business?

A used car, a pre-owned vehicle, or a secondhand car, is a vehicle that has previously had one or more retail owners. Used cars are sold through a variety of outlets, including franchise and independent car dealers, rental car companies, buy here pay dealerships, leasing offices, auctions, and private party sales. Some car retailers offer "no-haggle prices," "certified" used cars, and extended service plans or warranties.

India used car market was valued at USD 27 billion in 2020, and it is expected to reach USD 50 billion by 20206, registering a CAGR of 15% during the forecast period, 2021-2026.

The COVID-19 pandemic had a minimal impact on the industry. With the increased number of people preferring individual mobility and more finance options infused into the used car market, the market is set to grow considerably. Reduced cash inflow due to the pandemic has forced buyers to look for alternatives other than new cars, and the used car industry has great growth potential in these terms. With the sales and production of new vehicles hindered due to the pandemic, the immediate option for the buyers is the used car market.

The used car market evolved in the country, with the growth of the organized and semi-organized sales sectors. The pre-owned car market recorded sales of 4.4 million units in FY2020 as compared to only 2.8 million units of new passenger vehicles in the same year.

Motivation for the Problem Undertaken

The used car market is one of the biggest industries in India. Such a great industry also has very frequent changes. Price prediction is one of the challenging tasks both in real life and in machine learning. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies, and focusing on changing trends in sales and purchases. This certainly requires an effective and efficient machine learning model to be developed which can be a 'Star in the Dark Night' for the businesses.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

Our objective for the given business problem is to identify the price of the used car. For the task to be done, we are required to fetch used car data from various used car websites.

We extracted data of around 7000 cars. Details we fetched are Brand name, Model Name, Variant, Transmission type, fuel type, kms driven, and owner number.

During our analysis, we observe that the car that is 3-6 years old are preferred and are most likely to be sold quickly. Usually, cars are mid-range between 500k to 2000k. Although, we saw an increasing trend towards luxury cars and bigger brands.

Data Sources and their formats

For the given problem we collected the data from various used car websites like Cardekho.com, Carwale.com, Cars24.com. We scraped details about the car which are relevant to buy or sell one. We then create a final dataset that was used to train the machine learning model.

We used Selenium web driver to scrape data and defined functions to scrape data from various websites:

```
# Creating a function to extract data from Cars24.com
def cars24_list(url):

    # Establishing Connection with the website
    driver.get(url_c24)
    time.sleep(2)

    # Closing the location bar is appeared
    try:
        driver.find_element_by_xpath("//div[@class='float-right TBxAN']").click()
    except NoSuchElementException:
        pass

    # Clicking on Buy used car tab which is the 3rd option on the header
    driver.find_element_by_xpath("//ul[@class='float-right _1yq3e']/li[3]").click()
    time.sleep(2)

    # Creating a list to store urls
    urls=[]

    # Creating a for loop to iterate through cities to fetch the data
```

```
# Clicking one city at a time
driver.find_element_by_xpath(f"//ul[@class='_16Bvy']/li[{i}]").click()
time.sleep(2)


# Defining a function to iterate through bodytype
for fltr in range(1,6):

    # Clicking More Filter button
    driver.find_element_by_xpath("//ul[@class='_1bWR_']/li[6]/div").click()
    time.sleep(1)

    # Selecting a bodytype
    driver.find_element_by_xpath(f"//div[@class='dJM1f']/div[3]/ul/li[{fltr}]").click()
    time.sleep(1)

    # Cliking SHOW CARS button
    driver.find_element_by_xpath("//button[@class='btn _3BmFU AgO6q']").click()
    time.sleep(2)

    # Scrolling to get more cars in view
    for x in range(0,12):
        driver.execute_script('window.scrollBy(0,1000)')
        time.sleep(1)

    # Collecting link for each bodytype for each city
```

Function to get data from Cars24.com.

```
# Storing URL in a variable
url_cardekho=('https://www.cardekho.com/')
```

```
# Creating a function to extract data from Cardekho.com
def cardekho_list(url):

    # Establishing Connection with the website
    driver.get(url)
    time.sleep(2)

    # Creating an empty list to store URLs
    urls=[]

    # Creating a for loop to iterate through the list of cities
    for x in range(1,13):
        try:
            # Hovering over the Used Car option and hovering cars in your city and then selecting first city

            driver.find_element_by_xpath("//nav[@class='gsc_col-sm-12 gsc_col-md-12']/ul/li[2]").click()
            time.sleep(1)

            #Clicking on the location input field

            driver.find_element_by_xpath("//div[@class='inputfield']").click()
            time.sleep(1)

            # Selecting city one by one

            driver.find_element_by_xpath(f"//ul[@class='listing gsc_col-md-12 gsc_col-lg-12']/li[{x}]").click()
            time.sleep(2)
    # Closing location bar if available
    driver.find_element_by_xpath("//div[@class='gsc_modal gsc_modalWindow gsc_container show LocationEditScreen gsc_thin_
except NoSuchElementException:
    pass

# Creating a for loop to iterate through bodytype for each city

for i in range(1,7):

    driver.find_element_by_xpath(f"//div[@class='outBox bodyTypefilter bodytypeby']/ul/li[{i}]").click()
    time.sleep(3)

    for x in range(0,5):
        driver.execute_script('window.scrollBy(0,1000)')
        time.sleep(1)

    # Collecting URLs for the cars
    for link in driver.find_elements_by_xpath("//div[@class='holder']/div[1]/div[1]/a"):
        urls.append(link.get_attribute('href'))

    time.sleep(3)

    driver.find_element_by_xpath(f"//div[@class='outBox bodyTypefilter bodytypeby']/ul/li[{i}]").click()
    time.sleep(3)
```

```
# Creating a for loop to iterate through URLs to collect cars data
for url in urls:
    driver.get(url)
    time.sleep(3)
    try:
        try:
            cardekho_year.append(driver.find_element_by_xpath("//div[@class='paddingBorder clearfix']/div[1]/h1").text.split(

            cardekho_brand.append(driver.find_element_by_xpath("//div[@class='paddingBorder clearfix']/div[1]/h1").text.split

            name=driver.find_element_by_xpath("//div[@class='paddingBorder clearfix']/div[1]/h1").text.split()[2:4]
            cardekho_name.append(' '.join(name))

            cardekho_variant.append(driver.find_element_by_xpath("//div[@class='paddingBorder clearfix']/div[1]/div[1]").text

            cardekho_price.append(driver.find_element_by_xpath("//div[@class='priceSection']/span[2]").text.replace(',',''))

            cardekho_transmission.append(driver.find_element_by_xpath("//ul[@class='gsc_row detailsList'][1]/li[8]/div/div[2]

            cardekho_kms.append(driver.find_element_by_xpath("//ul[@class='gsc_row detailsList'][1]/li[4]/div/div[2]").text.r

            cardekho_fuel.append(driver.find_element_by_xpath("//ul[@class='gsc_row detailsList'][1]/li[3]/div/div[2]").text)

            cardekho_owner.append(driver.find_element_by_xpath("//ul[@class='gsc_row detailsList'][1]/li[6]/div/div[2]").text

            driver.execute_script('window.scrollTo(0,document.body.scrollHeight)')
            time.sleep(5)

            cardekho_location.append(driver.find_element_by_xpath("//div[@class='recomendedcar-Sec']/h2").text.split()[-1])
```

Function for Cardekho.com

```
# Storing URL in a variable
carwale_url=('https://www.carwale.com/')
```

```
# Creating a function to extract data from Carwale.com
def carwale_list(url):

    # Establishing connection with the website
    driver.get(url)
    time.sleep(3)

    # Hover over 'Used Car' tab and click 'find used cars' option
    actions.move_to_element(driver.find_element_by_xpath("//nav[@class='o-fzokld o-fzoker o-cpnuEd o-bUVylL o-cqgkZn o-eoatGj o-b
    time.sleep(1)
    driver.find_element_by_xpath("//ul[@class='o-cpnuEd o-XylGE _1fftdA']/a[1]").click()
    time.sleep(3)


    # Closing Locaion bar if visible
    try:
        driver.find_element_by_id('closeLocIcon').click()
    except NoSuchElementException:
        pass

    # Clicking on used button through breadcrumb
    driver.find_element_by_xpath("//ul[@id='breadCrumb']/li[2]/a").click()
    time.sleep(3)

    # Scrolling to view cities list
    for i in range(0,2):
        driver.execute_script('window.scrollBy(0,1000)')
        time.sleep(1)
```

```
    time.sleep(1)

# Storing City page links in a list
city_links=[]

for l in driver.find_elements_by_xpath("//div[@class='text-center']/a"):
    city_links.append(l.get_attribute('href'))

# Defining a list of bodytype
body=['Sedan','Hatchback','SUV/MUV','Minivan/Van','Coupe']

# Create an empty list to store URLs
urls=[]

# Iterating through a list of cities
for city in city_links:
    driver.get(city)
    time.sleep(3)

    # Iterating through each bodytype
    for bdy in body:

        # Move to the top of page
        driver.find_element_by_tag_name('body').send_keys(Keys.HOME)
        time.sleep(2)
```

We then collected all datasets and combine them to form the one which will be used. Here is a glimse of dataset:

Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
Maruti	S PRESSO	VXI	NaN	2020	179	Petrol	1st Owner	AHMEDABAD	434399
Maruti	Baleno DELTA	DELTA	MANUAL	2020	3920	Petrol	1st Owner	AHMEDABAD	652799
Maruti	S PRESSO	VXI	NaN	2020	3176	Petrol	1st Owner	AHMEDABAD	402199
Hyundai	Grand i10	MAGNA	MANUAL	2014	15033	Petrol	1st Owner	AHMEDABAD	372099
Maruti	Alto 800	VXI	MANUAL	2014	12535	Petrol	1st Owner	AHMEDABAD	298899
Maruti	Swift ZXI	ZXI	MANUAL	2012	15538	Petrol	1st Owner	AHMEDABAD	449099
Maruti	Baleno SIGMA	SIGMA	MANUAL	2020	5645	Petrol	1st Owner	AHMEDABAD	568999
Hyundai	Grand i10	ASTA	AUTOMATIC	2014	26779	Petrol	1st Owner	AHMEDABAD	420000
Maruti	Alto 800	LXI	MANUAL	2017	9527	Petrol	1st Owner	AHMEDABAD	304999
Maruti	Alto 800	LXI	MANUAL	2015	9217	Petrol	1st Owner	AHMEDABAD	287899
Hyundai	Grand i10	MAGNA	MANUAL	2017	9249	Petrol	1st Owner	AHMEDABAD	424399
Maruti	Swift VXI	VXI	MANUAL	2014	34710	Petrol	2nd Owner	AHMEDABAD	374699
Hyundai	Grand i10	SPORTS	MANUAL	2014	22836	Petrol	1st Owner	AHMEDABAD	383999
Honda	Brio 1.2	1.2	AUTOMATIC	2015	22774	Petrol	1st Owner	AHMEDABAD	430599
Renault	Kwid RXL	RXL	MANUAL	2019	4037	Petrol	1st Owner	AHMEDABAD	329099
Maruti	S PRESSO	VXI	MANUAL	2020	7556	Petrol	1st Owner	AHMEDABAD	412199

For the given dataset we have 10 columns in total out of which 7 are categorical and 3 are integer types including our target variable Price. Given below is the overviews of the dataset.

```
# Checking Dataset Info

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6941 entries, 0 to 6940
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         6941 non-null   int64
 1   Brand              6941 non-null   object
 2   Model              6941 non-null   object
 3   Variant            6941 non-null   object
 4   Transmission_Type  6804 non-null   object
 5   Year_of_Make       6941 non-null   object
 6   Kms_Driven         6941 non-null   object
 7   Fuel_Type          6941 non-null   object
 8   Owner              6941 non-null   object
 9   Location           6916 non-null   object
 10  Price              6941 non-null   object
dtypes: int64(1), object(10)
```

Given above is the information about the features. We have null values in the dataset. Given the table below displays the null count and the data types.

```
# Checking datatypes
df.dtypes
```

```
Unnamed: 0            int64
Brand                object
Model                object
Variant              object
Transmission_Type    object
Year_of_Make         object
Kms_Driven           object
Fuel_Type            object
Owner                object
Location             object
Price                object
dtype: object
```

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

```
Unnamed: 0             0
Brand                  0
Model                  0
Variant                0
Transmission_Type    137
Year_of_Make           0
Kms_Driven             0
Fuel_Type              0
Owner                  0
Location              25
Price                  0
```

Data Correction and Cleaning Done

dropping irrelevant column

```
df=df.drop(['Unnamed: 0'],axis=1)
```

Treating Null Values

```
df=df.dropna()
```

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

```
Unnamed: 0             0
Brand                  0
Model                  0
Variant                0
Transmission_Type    137
Year_of_Make           0
Kms_Driven             0
Fuel_Type              0
Owner                  0
Location              25
Price                  0
dtype: int64
```

We dropped the null values. Checking dataset again

```
df.shape
```

```
(6780, 10)
```

During data scraping we appended not found values with a '-'. We check if there is any such entries and remove it

```
df[df['Brand']=='-'].head(10)
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
495	-	-	-	-	-	-	-	-	-	-
517	-	-	-	-	-	-	-	-	-	-
524	-	-	-	-	-	-	-	-	-	-
534	-	-	-	-	-	-	-	-	-	-
541	-	-	-	-	-	-	-	-	-	-
574	-	-	-	-	-	-	-	-	-	-
583	-	-	-	-	-	-	-	-	-	-
634	-	-	-	-	-	-	-	-	-	-
669	-	-	-	-	-	-	-	-	-	-
700	-	-	-	-	-	-	-	-	-	-

We have several rows with Dashed entry. We wil drop these rows.

```
df[df['Price']=='-']
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
5976	Ferrari	488	GTB	Automatic	2017	3000	Petrol	First	Delhi	-
5979	Lamborghini	Huracan	LP 610-4	Automatic	2014	16000	Petrol	Third	Faridabad	-
5981	Bentley	Continental GT	Speed	Automatic	2012	23000	Petrol	First	Delhi	-
5996	Rolls-Royce	Phantom	EWB	Automatic	2008	15632	Petrol	First	Mumbai	-
6098	Maruti Suzuki	Omni	8 STR BS-II	Manual	2005	25000	Petrol	First	Mumbai	-
6109	Bentley	Continental GT	Coupe	Automatic	2012	30000	Petrol	Second	Mumbai	-
6112	Lamborghini	Aventador	LP 700-4	Automatic	2012	11000	Petrol	Second	Pune	-
6125	BMW	i8	1.5 Hybrid	Automatic	2016	11000	Petrol	First	Mumbai	-
6396	Ford	Ikon	1.6 EXi	Manual	2005	75582	Petrol	Second	Hyderabad	-
6602	BMW	i8	1.5 Hybrid	Automatic	2019	1300	Petrol	First	Kolkata	-
6621	Mercedes-Benz	S-Class	S 400	Automatic	2021	1000	Petrol	First	Pune	-
6687	BMW	X3 M	M	Automatic	2019	7215	Petrol	UnRegistered Car	Pune	-
6717	Lamborghini	Aventador	LP 700-4	Automatic	2012	11000	Petrol	Second	Pune	-
6720	Bentley	Continental GT	Coupe	Automatic	2011	10000	Petrol	Second	Pune	-
6723	Bentley	Continental GT	Coupe	Automatic	2012	30000	Petrol	Second	Pune	-
6726	Audi	R8	5.2 V10 Plus	Automatic	2016	60000	Petrol + Petrol	First	Pune	-
6934	Lamborghini	Huracan	LP 610-4	Automatic	2014	16000	Petrol	Third	Faridabad	-

We also have some dashed entries in our Traget Variable which needs to be removed

```
df.drop(df[df['Price']=='-'].index,inplace=True)
```

Converting numerical columns to integer

```
df['Year_of_Make']=df.Year_of_Make.astype('int')
df['Kms_Driven']=df.Kms_Driven.astype('int')
df['Price']=df.Price.astype('int')
```

```
df.dtypes
```

```
Brand                object
Model                object
Variant              object
Transmission_Type    object
Year_of_Make          int32
Kms_Driven            int32
Fuel_Type            object
Owner                object
Location             object
Price                 int32
```

We explored each column one by one and removed duplicates and errored values.

Checking the value count of less diverse columns first and treating them

```
df.Transmission_Type.value_counts()
```

```
Manual       2693
MANUAL       2072
Automatic    1407
AUTOMATIC     348
UP14            8
UP16            7
DL9C            6
DL7C            6
HR26            4
DL10            4
HR87            4
MH01            3
DL3C            2
DL14            2
DL2C            2
MH12            2
MH03            2
TS08            2
DL6C            2
KA05            1
```

we have duplicate as well as errored values. We treat duplicate values and drop the errored values

```
df.drop(df[df['Transmission_Type']=='UP14'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='UP16'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL9C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL7C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='HR26'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL10'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='HR87'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH01'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL3C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL14'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL2C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH12'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH03'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='TS08'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL6C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA05'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH11'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='AP36'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA51'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA50'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA02'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH02'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA03'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='TS10'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA04'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH13'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='DL4C'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='MH14'].index,inplace=True)
df.drop(df[df['Transmission_Type']=='KA53'].index,inplace=True)
```

```
df.Fuel_Type.value_counts()
```

```
Diesel              3159
Petrol              3150
CNG                   91
Petrol + CNG          49
Petrol + Petrol       23
CNG + Cng             18
Petrol + Cng           9
Petrol + LPG           5
CNG + CNG              3
Hybrid                 2
LPG                    2
CNG + Petrol           2
Petrol + Lpg           1
LPG + Lpg              1
Electric               1
CNG + Petrol+Cng       1
Petrol + Diesel        1
Diesel + Diesel        1
LPG + LPG              1
Name: Fuel_Type, dtype: int64
```

We have duplicate entries here which we will replace

```
df.replace({'Fuel_Type':{'Petrol + Petrol':'Petrol', 'CNG + Cng':'CNG', 'Petrol + Cng':'Petrol + CNG', 'CNG + CNG':'CNG', 'CNG +
```

```
df.Fuel_Type.value_counts()
```

```
Petrol             3173
Diesel             3160
CNG                 112
Petrol + CNG         61
Petrol + LPG          6
LPG                   4
Hybrid                2
Electric              1
Petrol + Diesel       1
Name: Fuel_Type, dtype: int64
```

We removed the duplicates. We observe an entry which is Petrol + Diesel which is not possible. Thus, we assume it to be an error and thus drop this entry

```
df.drop(df[df['Fuel_Type']=='Petrol + Diesel'].index,inplace=True)
```

Moving on to Owner Column

```
df.Owner.value_counts()
```

```
1st Owner              2608
First Owner            1652
First                   829
2nd Owner               727
Second Owner            364
Second                  177
3rd Owner                76
Third Owner              38
UnRegistered Car         17
Third                    15
Fourth & Above Owner      6
4th Owner                 5
Test Drive Car            3
4 or More                 2
Name: Owner, dtype: int64
```

Observed duplicate entries and will treat it

```
df.replace({'Owner':{'1st Owner': 1, 'First Owner':1, 'First':1, '2nd Owner':2, 'Second Owner':2, 'Second':2, '3rd Owner':3, 'Thi
```

```
df.Owner.value_counts()
```

```
1    5089
2    1268
3     129
0      20
4      13
Name: Owner, dtype: int64
```

Moving on to Location column

```
df.Location.value_counts()
```

```
Delhi          924
AHMEDABAD      620
Ahmedabad      445
Pune           425
Chennai        425
Bangalore      423
Mumbai         404
PUNE           374
BENGALURU      365
DELHI          340
CHENNAI        333
Hyderabad      329
Kolkata        269
Gurgaon        181
HYDERABAD      164
MUMBAI         100
Noida           90
Chandigarh      80
NOIDA           54
Thane           41
```

Here also we observe duplicates. Duplication is because of uppercase and lowercase characters. Thus, we convert the text in the column to lowercase and replace Bangalore with Bangaluru (official name).

```
df['Location']=df['Location'].str.lower()
```

```
df.replace({'Location':{'bangalore': 'bengaluru'}},inplace=True)
```

```
df.Location.value_counts()
```

```
delhi          1264
ahmedabad      1065
pune            799
bengaluru       788
chennai         758
mumbai          504
hyderabad       493
kolkata         269
gurgaon         212
noida           144
chandigarh       80
faridabad        50
thane            41
ghaziabad        24
navi mumbai      14
surat            10
dehradun          2
mohali            1
chinchwad         1
Name: Location, dtype: int64
```

We have single entry for Mohali and Chinchwad. Thus, we will replace chinchwad with nearest city Pune and mohali with Chandigarh.

```
df.replace({'Location':{'mohali':'chandigarh','chinchwad':'pune'}},inplace=True)
```

```
df.Location.value_counts()
```

```
delhi          1264
ahmedabad      1065
pune            800
bengaluru       788
chennai         758
mumbai          504
hyderabad       493
kolkata         269
gurgaon         212
noida           144
chandigarh       81
faridabad        50
thane            41
ghaziabad        24
navi mumbai      14
surat            10
dehradun          2
Name: Location, dtype: int64
```

Now the location column is cleaned lets move to Brand name.

```
df.Brand.value_counts()
```

```
Maruti          2162
Hyundai          784
Toyota           621
Honda            422
Maruti Suzuki    313
Mahindra         300
BMW              285
Mercedes-Benz    251
Audi             241
Renault          188
Ford             176
Volkswagen       158
Tata              87
Skoda             84
Nissan            55
Mercedes          52
Land              40
Jeep              39
Chevrolet         39
Jaguar            38
```

First we will convert text to lowercase to eliminate case differences and replace mercedes with mercede-benz, aston with aston martin, land and landrover with land rover.

```
df['Brand']=df['Brand'].str.lower()
```

```
df.replace({'Brand':{'mercedes':'mercedes-benz', 'land':'land rover','landrover':'land rover', 'aston':'aston martin'}},inplace=T
```

Moving to Model Name first we will lower the case and see the value counts

```
df['Model']=df['Model'].str.lower()
```

```
df.Model.value_counts()
```

```
swift dzire          459
vitara brezza        211
eeco 5               173
innova 2.5           172
grand i10            163
alto 800             152
eeco                 148
innova crysta        119
wagon r              115
alto k10             100
ciaz                  95
corolla altis         91
baleno                80
creta 1.6             78
ecosport 1.5          76
swift                 71
city                  69
3 series              65
alto lxi              54
5 series              54
```

We will use the model column to examine the Variant column in small pieces(like below) to make the data in Variant column less diverse and replace any logically repetitive entries. Also, we will replace repetitive entries in the Model column encountered along.

We used most occurring models to check the occurrence and correct the variant and model name and reduce the diversity as much as possible.

```
df[df['Model']==('swift dzire')]
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
80	maruti	swift dzire	vdi	Manual	2012	48390	Diesel	1	ahmedabad	311499
81	maruti	swift dzire	zdi	Manual	2014	27735	Diesel	1	ahmedabad	454899
82	maruti	swift dzire	vdi	Manual	2013	39947	Diesel	1	ahmedabad	393299
83	maruti	swift dzire	vdi	Manual	2013	44284	Diesel	1	ahmedabad	376999
85	maruti	swift dzire	vdi	Manual	2012	42478	Diesel	2	ahmedabad	345599
90	maruti	swift dzire	vxi	Manual	2010	44518	Petrol	1	ahmedabad	264299
93	maruti	swift dzire	vdi	Manual	2014	42045	Diesel	1	ahmedabad	458199
95	maruti	swift dzire	vdi	Manual	2012	65946	Diesel	1	ahmedabad	329299
96	maruti	swift dzire	vxi	Manual	2013	27712	Petrol	1	ahmedabad	385000
98	maruti	swift dzire	vdi	Manual	2012	82064	Diesel	1	ahmedabad	346799
99	maruti	swift dzire	vdi	Manual	2012	61164	Diesel	1	ahmedabad	320599
100	maruti	swift dzire	zdi	Manual	2012	79396	Diesel	1	ahmedabad	365899

```
# checking unique value count in Variant and Model
print('Number of unique variables in model column is : ',df.Model.nunique())
print('Number of unique variables in variant column is : ',df.Variant.nunique())
```

```
Number of unique variables in model column is :  700
Number of unique variables in variant column is :  1284
```

```
# converting the variant column to lower case to eliminate any case differences
df['Variant']=df['Variant'].str.lower()
```

```
# Checking dataset where model column has swift in name
df2[df2['Model'].str.contains('swift')]
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
5	maruti	swift zxi	zxi	Manual	2012	15538	Petrol	1	ahmedabad	449099
11	maruti	swift vxi	vxi	Manual	2014	34710	Petrol	2	ahmedabad	374699
17	maruti	swift lxi	lxi	Manual	2019	7032	Petrol	1	ahmedabad	509399
18	maruti	swift lxi	lxi	Manual	2012	40551	Petrol	1	ahmedabad	314599
21	maruti	swift vdi	vdi	Manual	2013	39124	Diesel	1	ahmedabad	312499
32	maruti	swift zdi	zdi	Manual	2013	24022	Diesel	1	ahmedabad	409299
41	maruti	swift vdi	vdi	Manual	2017	16646	Diesel	1	ahmedabad	479299
45	maruti	swift lxi	lxi	Manual	2013	26908	Petrol	1	ahmedabad	394599
47	maruti	swift lxi	lxi	Manual	2017	24562	Petrol	1	ahmedabad	438699
54	maruti	swift vxi	vxi	Manual	2012	34092	Petrol	1	ahmedabad	338999
62	maruti	swift lxi	lxi	Manual	2019	10390	Petrol	1	ahmedabad	549399
77	maruti	swift vdi	vdi	Manual	2012	47140	Diesel	1	ahmedabad	338999

we fetched rows which contains swift in model name to a new dataframe to get an idea of values

```
test1=df2[df2['Model'].str.contains('swift')]
```

```
test1.Variant.value_counts()
```

```
vxi                195
vdi                166
zdi                 42
zxi                 35
ldi                 29
lxi                 27
vxi 1.2 bs iv       17
vxi 1.2             16
```

```
# First we replace Brand name from Maruti to Maruti Suzuki where model name has swift in it

df2.loc[(df2['Model'].str.contains('swift')),'Brand']='maruti suzuki'
```

```
df2[df2['Model'].str.contains('swift')]
```

32	maruti suzuki	swift zdi	zdi	Manual	2013	24022	Diesel	1	ahmedabad	409299
41	maruti suzuki	swift vdi	vdi	Manual	2017	16646	Diesel	1	ahmedabad	479299
45	maruti suzuki	swift lxi	lxi	Manual	2013	26908	Petrol	1	ahmedabad	394599
47	maruti suzuki	swift lxi	lxi	Manual	2017	24562	Petrol	1	ahmedabad	438699
54	maruti suzuki	swift vxi	vxi	Manual	2012	34092	Petrol	1	ahmedabad	338999
62	maruti suzuki	swift lxi	lxi	Manual	2019	10390	Petrol	1	ahmedabad	549399
77	maruti suzuki	swift vdi	vdi	Manual	2012	47140	Diesel	1	ahmedabad	338999
80	maruti suzuki	swift dzire	vdi	Manual	2012	48390	Diesel	1	ahmedabad	311499
81	maruti suzuki	swift dzire	zdi	Manual	2014	27735	Diesel	1	ahmedabad	454899
82	maruti suzuki	swift dzire	vdi	Manual	2013	39947	Diesel	1	ahmedabad	393299
83	maruti suzuki	swift dzire	vdi	Manual	2013	44284	Diesel	1	ahmedabad	376999
85	maruti suzuki	swift dzire	vdi	Manual	2012	42478	Diesel	2	ahmedabad	345599
90	maruti suzuki	swift dzire	vxi	Manual	2010	44518	Petrol	1	ahmedabad	264299

Replacing swift vxi, swift zdi etc in Model column with swift and variants with variants removing garbage text.

```
df2.replace({'Model':{'swift vxi':'swift','swift lxi':'swift', 'swift vdi':'swift', 'swift ldi':'swift', 'swift zxi':'swift', 'sw
```

```
# Now we will replace values in variant

df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('vxi')),'Variant']='vxi'
df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('lxi')),'Variant']='lxi'
df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('vdi')),'Variant']='vdi'
df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('ldi')),'Variant']='ldi'
df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('zxi')),'Variant']='zxi'
df2.loc[(df2['Model'].str.contains('swift')) & (df2['Variant'].str.contains('zdi')),'Variant']='zdi'
```

```
test1=df2[df2['Model'].str.contains('swift')]
```

```
test1.Model.value_counts()
```

```
swift dzire    459
swift          148
Name: Model, dtype: int64
```

```
test1=df2[df2['Brand'].str.contains('audi')]
```

```
test1
```

258	audi	a6 2.0	2.0	Automatic	2013	107465	Diesel	2	ahmedabad	1329199
263	audi	a4 2.0	2.0	Automatic	2013	44762	Diesel	2	ahmedabad	1072099
264	audi	a4 2.0	2.0	Automatic	2012	90722	Diesel	2	ahmedabad	910999
269	audi	a4 35	35	Automatic	2015	47680	Diesel	1	ahmedabad	1739799
279	audi	q3 35	35	Automatic	2015	13265	Diesel	1	ahmedabad	2350000
293	audi	q3 30	30	Manual	2015	69074	Diesel	1	ahmedabad	1496499
309	audi	q3 2.0	2.0	Automatic	2013	114155	Diesel	2	ahmedabad	1127899
325	audi	q3 35	35	Automatic	2015	67720	Diesel	3	ahmedabad	1540599
327	audi	q3 35	35	Automatic	2015	83813	Diesel	2	ahmedabad	1506599
335	audi	q3 2.0	2.0	Automatic	2012	80021	Diesel	2	ahmedabad	923299
577	audi	a4 2.0	2.0	Automatic	2012	45643	Diesel	1	delhi	1242099
596	audi	a6 2.0	2.0	Automatic	2013	107465	Diesel	2	delhi	1329199
601	audi	a4 2.0	2.0	Automatic	2013	44762	Diesel	2	delhi	1072099

```
test1.Model.value_counts()
```

```
q7 2006-2020    21
a4 2.0          19
a6 2.0          19
a6 35           16
q3 35           15
q3 2.0          15
q5 2.0          12
q3 30           11
q8 55            8
```

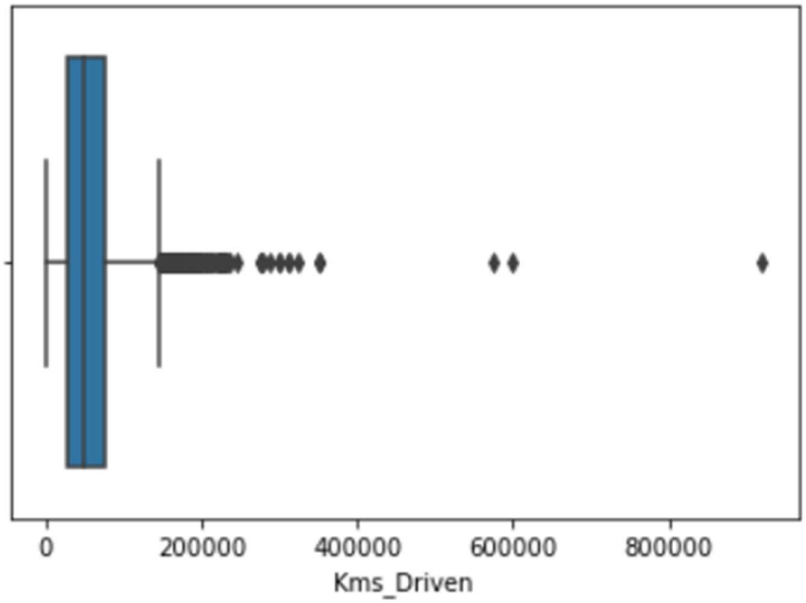
```
# Correcting Variants
df2.loc[(df2['Model']==('a3 35tdi')) & (df2['Variant']=='35tdi'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('a4 2.0')) & (df2['Variant']=='tdi'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('q7 2006-2020')) & (df2['Variant']=='35tdi'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('a3 35tdi')) & (df2['Variant']=='35tdi'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('a4 2.0')) & (df2['Variant']=='2.0'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('a6 2.0')) & (df2['Variant']=='2.0'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('a4 35')) & (df2['Variant']=='35'),'Variant']='35 tdi'

df2.loc[(df2['Model']==('q3 35')) & (df2['Variant']=='35'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('q3 30')) & (df2['Variant']=='30'),'Variant']='30 tdi'
df2.loc[(df2['Model']==('q3 2.0')) & (df2['Variant']=='2.0'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('a4 2.0')) & (df2['Variant']=='tdi multitronic'),'Variant']='2.0 tdi multitronic'
df2.loc[(df2['Model']==('a4 2.0')) & (df2['Variant']=='tdi 177'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('q5 30')) & (df2['Variant']=='tdi quattro'),'Variant']='30 tdi quattro'

df2.loc[(df2['Model']==('q3 2.0')) & (df2['Variant']=='tdi quattro'),'Variant']='2.0 tdi quattro'
df2.loc[(df2['Model']==('a6 35')) & (df2['Variant']=='tdi'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('q5 2.0')) & (df2['Variant']=='tdi'),'Variant']='2.0 tdi'
df2.loc[(df2['Model']==('a6 2.0')) & (df2['Variant']=='tdi'),'Variant']='2.0 tdi'

df2.loc[(df2['Model']==('a3 40')) & (df2['Variant']=='40 tfsi'),'Variant']='40 tfsi premium'
df2.loc[(df2['Model']==('a6 35')) & (df2['Variant']=='35'),'Variant']='35 tdi'
df2.loc[(df2['Model']==('q3 35')) & (df2['Variant']=='tdi quattro'),'Variant']='35 tdi quattro'
df2.loc[(df2['Model']==('q5 3.0')) & (df2['Variant']=='tdi'),'Variant']='3.0 tdi'
df2.loc[(df2['Model']==('q5 3.0')) & (df2['Variant']=='tdi quattro'),'Variant']='3.0 tdi quattro'

df2.loc[(df2['Model']==('q5 2.0')) & (df2['Variant']=='tdi premium'),'Variant']='2.0 tdi premium'
df2.loc[(df2['Model']==('a6 2.0')) & (df2['Variant']=='tdi premium'),'Variant']='2.0 tdi premium'
df2.loc[(df2['Model']==('q5 30')) & (df2['Variant']=='30 tdi quattro'),'Variant']='30 tdi quattro'


df2.loc[(df2['Model']==('a4 35')) & (df2['Variant']=='tdi premium'),'Variant']='35 tdi premium'
df2.loc[(df2['Model']==('q5 35tdi')) & (df2['Variant']=='premium plus'),'Variant']='35 tdi premium'
df2.loc[(df2['Model']==('q3 30')) & (df2['Variant']=='tdi s'),'Variant']='30 tdi'

df2.loc[(df2['Model']==('q3 2.0')) & (df2['Variant']=='2.0 tdi quattro'),'Variant']='2.0 tdi quattro'
df2.loc[(df2['Model']==('a6 2.8')) & (df2['Variant']=='fsi'),'Variant']='2.8 fsi'
```

Similarly, we checked each brand and model to make the data appropriate for machine learning.

Data Preprocessing Done

Outlying Values

Talking about the distribution, we observe that there are some values in the kms_driven column which act as outlying values. On examining we found the entry to errored. The make of the car was 2 years back and it records 920k kms. This certainly doesn't seem right. So we just dropped the vale.



Lets examine the outlying entrie in the Kms_Driven column

```
df3[df3['Kms_Driven']>400000]
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location	Price
5487	honda	city	v	Automatic	2016	600000	Petrol	2	delhi	675000
6386	nissan	terrano	xl	Manual	2014	575000	Diesel	2	ahmedabad	575000
6441	maruti suzuki	wagon r	vxi(a)	Manual	2019	920000	Petrol	1	chandigarh	459999

Its hardly possible for a car to be driven 920000kms in 2 years. It might be an error entry thus we will drop it

```
df3=df3[df3['Kms_Driven']<600001]
```

Abnormal data distribution or Skewness

We observed the data distribution in the dataset. We listed the features for abnormality or skewness in the data in descending order. We observed features have skewness present which needs to be treated. AS the price is our target variable and the Year of making is categorical we have to correct skewness of the Kms_driven.

```
df3.skew().sort_values(ascending=False)
```

```
Price           3.535902
Kms_Driven      2.244174
Owner           1.895080
Year_of_Make   -0.773184
dtype: float64
```

Correcting skewness in kms driven column

```
#Importing Power Transform
from sklearn.preprocessing import power_transform
```

```
df3['Kms_Driven']=power_transform(df3['Kms_Driven'].values.reshape(-1,1))
```

```
df3.skew().sort_values(ascending=False)
```

```
Price           3.535902
Owner           1.895080
Kms_Driven      0.021872
Year_of_Make   -0.773184
dtype: float64
```

To remove the abnormality from the dataset we used Power Transformer. Power Transformer is an abnormality correction method in the Scikit-Learn's Preprocessing family. Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired. Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood. Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data. By default, zero-mean, unit-variance normalization is applied to the transformed data.

Encoding Categorical or Object Input Features

Encoding is the process of converting categorical or object-type data into an array of integers. The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are converted to ordinal integers. This results in a single column of integers (0 to n_categories - 1) per feature.

For Encoding we used Ordinal Encoder.

Encoding

```
: from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
```

```
: for i in x.columns:
    if x[i].dtypes=='O':
        x[i]=oe.fit_transform(x[i].values.reshape(-1,1))
```

	Brand	Model	Variant	Transmission_Type	Year_of_Make	Kms_Driven	Fuel_Type	Owner	Location
0	0.179936	-1.199793	-0.108383	0.604888	1.619359	-2.225153	0.990482	-0.485324	-1.215429
1	-0.912391	-0.189182	0.432322	0.604888	-0.338625	-1.285612	0.990482	-0.485324	-1.215429
2	0.179936	-1.316966	1.062468	0.604888	-0.338625	-1.442261	0.990482	-0.485324	-1.215429
3	0.179936	1.085067	1.334854	0.604888	-0.991287	-1.255953	0.990482	-0.485324	-1.215429
4	0.179936	-1.199793	0.765690	0.604888	1.619359	-2.015531	0.990482	-0.485324	-1.215429

Normalization or Scaling the Input Features

Data Normalization is a common practice in machine learning which consists of transforming numeric columns to a common scale. In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the learning process. However, it does not mean those variables are more important to predict the outcome of the model. Data normalization transforms multiscaled data to the same scale. After normalization, all variables have a similar influence on the model, improving the stability and performance of the learning algorithm.

For normalizing our input features we used Standard Scaler.

```
: from sklearn.preprocessing import StandardScaler
  sc=StandardScaler()
```

```
: x_new=sc.fit_transform(x)
  x=pd.DataFrame(x_new,columns=x.columns)
  x.head()
```

Hardware and Software Requirements and Tools Used

Software Requirement:

1. Anaconda Navigator software.

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

2. Jupyter Notebook - as an IDE for writing Python codes for analyzing data and machine learning purposes.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents that combine explanatory text, mathematics, computations, and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects

3. Python Programming

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and

code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed

4. Libraries Used

Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms basic linear algebra, basic statistical operations, random simulation and much more.

Pandas

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data-wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState's ActivePython.

Matplotlib

Matplotlib is a cross-platform, data visualization, and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

- The pyplot API is a hierarchy of Python code objects topped by *matplotlib.pyplot*

- An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

Selenium

Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. It provides extensions to emulate user interaction with browsers, a distribution server for scaling browser allocation, and the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers.

This project is made possible by volunteer contributors who have put in thousands of hours of their own time and made the source code freely available for anyone to use, enjoy, and improve. Selenium brings together browser vendors, engineers, and enthusiasts to further an open discussion around the automation of the web platform. The project organizes an annual conference to teach and nurture the community. At the core of Selenium is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers.

Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Scikit-Learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions,

encouraging academic and commercial use. Some popular groups of models provided by scikit-learn include:

- Clustering: for grouping unlabeled data such as KMeans.
- Cross-Validation: for estimating the performance of supervised models on unseen data.
- Datasets: for test datasets and for generating datasets with specific properties for investigating model behavior.
- Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization, and feature selection such as Principal component analysis.
- Ensemble methods: for combining the predictions of multiple supervised models.
- Feature extraction: for defining attributes in image and text data.
- Feature selection: for identifying meaningful attributes from which to create supervised models.
- Parameter Tuning: for getting the most out of supervised models.
- Manifold Learning: For summarizing and depicting complex multi-dimensional data.
- Supervised Models: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

Given scenario challenges us to predict the price of a used car. Predicting the prices is one of the favorite and challenging areas in the field of data science. For the task to be done, we scraped the required data from various websites with Price as our target variable. As our target variable is continuous we will move ahead with a regression approach for this problem.

Testing of Identified Approaches (Algorithms)

After performing various preprocessing tasks on our dataset like treating null values, distribution correction, encoding, and scaling in the dataset we made our data suitable for passing through the algorithms to train the best predicting model.

After preprocessing we created a train and test split using the input and the target variable. We divided the input features in the ratio where 33.33% of data goes for testing and 66.66% of data is used for training. Using one of the algorithms we tried to find the best random state to get the best results possible.

Finding best random state

```
maxAccu=0
maxRs=0
for i in range(1,500):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.33,random_state=i)
    dtr=DecisionTreeRegressor()
    dtr.fit(x_train,y_train)
    dtr_pred=dtr.predict(x_test)
    acc=r2_score(y_test,dtr_pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRs=i
print('Best R2 Score is : ', maxAccu, ' when Random state is : ',maxRs)
```

```
Best R2 Score is :  0.7294346985893442  when Random state is :  359
```

Using the above-stated random state we created our train and test data.

```
# Selecting random state
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.33,random_state=359)
```

With the train-test split done we move ahead and test our dataset for R2 score with various algorithms. Algorithms used for training are :

1. Linear Regression

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables, in particular, are significant predictors of the outcome variable, and in what way do they– indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables.

Linear regression is an attractive model because the representation is so simple. The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient, and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B0 + B1*x$$

where y = estimated dependent variable score, B0= constant, B1= regression coefficient, and x = score on the independent variable.

2. Support Vector Machine (Regressor)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection. The advantages of support vector machines are:

- Effective in high-dimensional spaces.
- Still effective in cases where the number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The method of Support Vector Classification can be extended to solve regression problems. This method is called Support Vector Regression. The model produced by support vector classification (as described above) depends only on a subset of the training data because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

3. Decision Tree Regressor

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are

usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.

- Able to handle multi-output problems.
- Uses a white-box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black-box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
-

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node, or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- Some concepts are hard to learn because decision trees do not express them easily, such as XOR, parity, or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset before fitting with the decision tree

Decision trees can also be applied to regression problems, using the DecisionTreeRegressor class.

4. KNeighbors Regressor

Sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods. Unsupervised nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering. Supervised

neighbors-based learning comes in two flavors: classification for data with discrete labels and regression for data with continuous labels.

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods since they simply "remember" all of their training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree).

Despite its simplicity, nearest neighbors have been successful in a large number of classification and regression problems, including handwritten digits and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors. Scikit-learn implements two different neighbors regressors: KNeighborsRegressor implements learning based on the nearest neighbors of each query point, where is an integer value specified by the user. RadiusNeighborsRegressor implements learning based on the neighbors within a fixed radius of the query point, where is a floating-point value specified by the user.

5. Ensemble Methods

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm to improve generalizability/robustness over a single estimator. Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

  Examples: Bagging methods, Forests of randomized trees,

- By contrast, in boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Examples: AdaBoost, Gradient Tree Boosting, …

❖ Random Forest Regressor

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features. (See the parameter tuning guidelines for more details).

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yields decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice, the variance reduction is often significant hence yielding an overall better model.

❖ Gradient Boosting Regressor

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.

GradientBoostingRegressor supports several different loss functions for regression which can be specified via the argument loss; the default loss function for regression is least squares ('ls').

❖ Ada Boost Regressor

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessings, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights, w_1 , w_2 …w_N, to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased

for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence [HTF].

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

This class implements the algorithm known as AdaBoost.R2

6. Extreme Gradient Boosting (XGB)

XGBoost stands for eXtreme Gradient Boosting. The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms.

It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.

XGBoost Features

The library is laser-focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer several advanced features.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

- Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
- Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
- Regularized Gradient Boosting with both L1 and L2 regularization.

System Features

The library provides a system for use in a range of computing environments, not least:

- Parallelization of tree construction using all of your CPU cores during training.

- Distributed Computing for training very large models using a cluster of machines.
- Out-of-Core Computing for very large datasets that don't fit into memory.
- Cache Optimization of data structures and algorithm to make the best use of hardware.

Algorithm Features

The implementation of the algorithm was engineered for the efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

- Sparse Aware implementation with automatic handling of missing data values.
- Block Structure to support the parallelization of tree construction.
- Continued Training so that you can further boost an already fitted model on new data.

7. Regularization Technique

This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, to avoid the risk of overfitting. A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).

$$Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$$

The fitting procedure involves a loss function, known as the residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function. Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

❖ Ridge Regression

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

The above image shows ridge regression, where the RSS is modified by adding the shrinkage quantity. Now, the coefficients are estimated by minimizing this function. Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model**.** The increase in flexibility of a model is represented by an increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high.

❖ Lasso Regression

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| \;=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

Lasso is another variation, in which the above function is minimized. This variation differs from ridge regression only in penalizing the high coefficients. It uses |βj|(modulus)instead of squares of β, as its penalty. In statistics, this is known as the L1 norm.

Figure: Importing required algorithms and evaluation metrics

```
# Importing Regression Algorithms & Metrics
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso,Ridge,ElasticNet
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

We created a For Loop to test the accuracy of all the algorithms.

```
# Creating For loop to print Training and Test R2 score
for m in model_list:
    model=m
    model.fit(x_train,y_train)
    model_pred_train=model.predict(x_train)
    model_pred=model.predict(x_test)
    print('Training Accuracy for the model ',m,' is: ',r2_score(y_train,model_pred_train)*100)
    print('Testing Accuracy for the model ',m,' is: ',r2_score(y_test,model_pred)*100)
    print('\n')
```

R2 scores:

```
Training Accuracy for the model  LinearRegression()  is:  38.702460021180876
Testing Accuracy for the model  LinearRegression()  is:  40.92322281532298


Training Accuracy for the model  SVR()  is:  -11.31517392885426
Testing Accuracy for the model  SVR()  is:  -11.779494395290913


Training Accuracy for the model  DecisionTreeRegressor()  is:  99.99967835233548
Testing Accuracy for the model  DecisionTreeRegressor()  is:  64.5410033135156


Training Accuracy for the model  KNeighborsRegressor()  is:  73.13932008093606
Testing Accuracy for the model  KNeighborsRegressor()  is:  65.82906931969839


Training Accuracy for the model  RandomForestRegressor()  is:  96.5376056121299
Testing Accuracy for the model  RandomForestRegressor()  is:  83.277838811492


Training Accuracy for the model  GradientBoostingRegressor()  is:  76.73881120907647
Testing Accuracy for the model  GradientBoostingRegressor()  is:  74.3953174999825


Training Accuracy for the model  AdaBoostRegressor()  is:  41.292885522104264
Testing Accuracy for the model  AdaBoostRegressor()  is:  44.966535905887774


Training Accuracy for the model  Lasso()  is:  38.70246002054103
Testing Accuracy for the model  Lasso()  is:  40.92320358120889


Training Accuracy for the model  Ridge()  is:  38.70245813679224
Testing Accuracy for the model  Ridge()  is:  40.921746800151226


Training Accuracy for the model  ElasticNet()  is:  34.89679242339104
Testing Accuracy for the model  ElasticNet()  is:  35.498981429871115


Training Accuracy for the model  XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=0)  is:  98.87789894608265
Testing Accuracy for the model  XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=0)  is:  84.24116877849315
```

We then Cross validated each algorithm to check for any overfitting or underfitting over various CV values.

```
# Linear Regression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr_pred=lr.predict(x_test)
testing_accu=r2_score(y_test,lr_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(lr,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 22.431666400034306 and the accuracy for testing is 40.92322281532298


At crossfold 3 the CV score is -17.232545331227247 and the accuracy for testing is 40.92322281532298


At crossfold 4 the CV score is 6.6949615932318185 and the accuracy for testing is 40.92322281532298


At crossfold 5 the CV score is -4.584059221405692 and the accuracy for testing is 40.92322281532298


At crossfold 6 the CV score is -9.395589570144697 and the accuracy for testing is 40.92322281532298


At crossfold 7 the CV score is 0.4638796756974539 and the accuracy for testing is 40.92322281532298


At crossfold 8 the CV score is -20.499131640613825 and the accuracy for testing is 40.92322281532298


At crossfold 9 the CV score is -7.186345363163535 and the accuracy for testing is 40.92322281532298
```

```
# DecisionTreeClassifier
dtr=DecisionTreeRegressor()
dtr.fit(x_train,y_train)
dtr_pred=dtr.predict(x_test)
testing_accu=r2_score(y_test,dtr_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(dtr,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 27.8783056553788 and the accuracy for testing is 69.11563268537093


At crossfold 3 the CV score is 14.829062982835998 and the accuracy for testing is 69.11563268537093


At crossfold 4 the CV score is 2.1585437965728427 and the accuracy for testing is 69.11563268537093


At crossfold 5 the CV score is 18.616737506282323 and the accuracy for testing is 69.11563268537093


At crossfold 6 the CV score is 22.066212685477836 and the accuracy for testing is 69.11563268537093


At crossfold 7 the CV score is 10.558577907085935 and the accuracy for testing is 69.11563268537093


At crossfold 8 the CV score is -35.0619309963071 and the accuracy for testing is 69.11563268537093


At crossfold 9 the CV score is 41.07406797223738 and the accuracy for testing is 69.11563268537093
```

```
# KNeighbor Regressor
knn=KNeighborsRegressor()
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
testing_accu=r2_score(y_test,knn_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(knn,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 30.464496185895385 and the accuracy for testing is 65.82906931969839


At crossfold 3 the CV score is 3.8072844553422933 and the accuracy for testing is 65.82906931969839


At crossfold 4 the CV score is 26.1330571881212 and the accuracy for testing is 65.82906931969839


At crossfold 5 the CV score is 26.48647309689312 and the accuracy for testing is 65.82906931969839


At crossfold 6 the CV score is 21.15780187603882 and the accuracy for testing is 65.82906931969839


At crossfold 7 the CV score is 24.79509056312244 and the accuracy for testing is 65.82906931969839


At crossfold 8 the CV score is 21.656414287784138 and the accuracy for testing is 65.82906931969839


At crossfold 9 the CV score is 34.93305836911589 and the accuracy for testing is 65.82906931969839
```

Ensemble methods

```
# RandomForestRegressor
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr_pred=rfr.predict(x_test)
testing_accu=r2_score(y_test,rfr_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(rfr,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 50.271079111530305 and the accuracy for testing is 83.86798830172366


At crossfold 3 the CV score is 49.602643205975426 and the accuracy for testing is 83.86798830172366


At crossfold 4 the CV score is 60.660902445590615 and the accuracy for testing is 83.86798830172366


At crossfold 5 the CV score is 62.28660835795182 and the accuracy for testing is 83.86798830172366


At crossfold 6 the CV score is 57.59761203429138 and the accuracy for testing is 83.86798830172366


At crossfold 7 the CV score is 59.47892355578504 and the accuracy for testing is 83.86798830172366


At crossfold 8 the CV score is 60.88672829063617 and the accuracy for testing is 83.86798830172366


At crossfold 9 the CV score is 70.7781355921376 and the accuracy for testing is 83.86798830172366
```

```
# Cross validating GradientBoostingRegressor

gbr = GradientBoostingRegressor()
gbr.fit(x_train, y_train)
gbr_pred=gbr.predict(x_test)
testing_accu=r2_score(y_test,gbr_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(gbr,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 53.98965639739538 and the accuracy for testing is 74.41227759087663


At crossfold 3 the CV score is 47.52980382935619 and the accuracy for testing is 74.41227759087663


At crossfold 4 the CV score is 55.70648790428257 and the accuracy for testing is 74.41227759087663


At crossfold 5 the CV score is 53.0353365337675 and the accuracy for testing is 74.41227759087663


At crossfold 6 the CV score is 51.490415942381915 and the accuracy for testing is 74.41227759087663


At crossfold 7 the CV score is 55.509289859008994 and the accuracy for testing is 74.41227759087663


At crossfold 8 the CV score is 52.21515514720117 and the accuracy for testing is 74.41227759087663


At crossfold 9 the CV score is 56.99307265365883 and the accuracy for testing is 74.41227759087663
```

```
# Cross validating Ada Boost Regressor

adb = AdaBoostRegressor()
adb.fit(x_train, y_train)
adb_pred=adb.predict(x_test)
testing_accu=r2_score(y_test,adb_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(adb,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 17.05948716290052 and the accuracy for testing is 40.74985146847815


At crossfold 3 the CV score is -69.06262058635924 and the accuracy for testing is 40.74985146847815


At crossfold 4 the CV score is -26.167879755041916 and the accuracy for testing is 40.74985146847815


At crossfold 5 the CV score is -53.5333874305041 and the accuracy for testing is 40.74985146847815


At crossfold 6 the CV score is -62.79131776542545 and the accuracy for testing is 40.74985146847815


At crossfold 7 the CV score is -19.708051089642872 and the accuracy for testing is 40.74985146847815


At crossfold 8 the CV score is -69.5465895047706 and the accuracy for testing is 40.74985146847815


At crossfold 9 the CV score is -52.22245287543763 and the accuracy for testing is 40.74985146847815
```

Extreme Gradient boosting

```
xgb = XGBRegressor(verbosity=0)
xgb.fit(x_train, y_train)
xgb_pred=xgb.predict(x_test)
testing_accu=r2_score(y_test,xgb_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(xgb,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')
```

```
At crossfold 2 the CV score is 57.8540483643537 and the accuracy for testing is 84.24116877849315


At crossfold 3 the CV score is 53.94856819878431 and the accuracy for testing is 84.24116877849315


At crossfold 4 the CV score is 64.58708885136011 and the accuracy for testing is 84.24116877849315


At crossfold 5 the CV score is 65.79083999258665 and the accuracy for testing is 84.24116877849315


At crossfold 6 the CV score is 61.11594005952541 and the accuracy for testing is 84.24116877849315


At crossfold 7 the CV score is 64.91761458609027 and the accuracy for testing is 84.24116877849315


At crossfold 8 the CV score is 63.936612779381896 and the accuracy for testing is 84.24116877849315


At crossfold 9 the CV score is 74.55769334455022 and the accuracy for testing is 84.24116877849315
```

Run and Evaluate selected models

Based on the R2 and cross-validation score, we selected top-performing algorithms. Extreme Gradient Boosting showed the highest R2 with minimum overfitting followed by Random Forest. We tested the selected models for Mean Absolute Error, and Root Mean Squared Error.

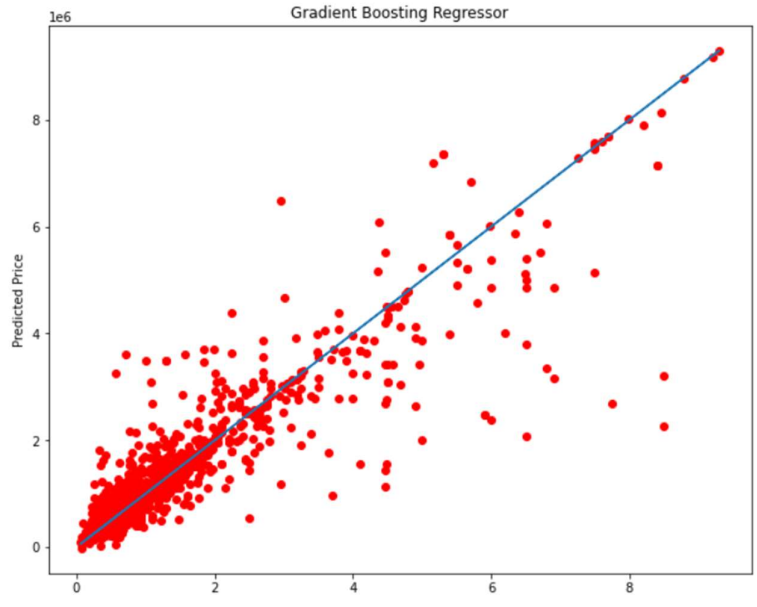
```
Random Forest Regression
Mean Squared Error of the model is  251667454321.6702
Mean Absolute Error of the model is  209721.91293880087
Root Mean Squared Error of the model is  501664.68315167475
-----------------------------------
Gradient Boosting Regression
Mean Squared Error of the model is  399181272679.1964
Mean Absolute Error of the model is  313069.54416096443
Root Mean Squared Error of the model is  631807.9397088932
-----------------------------------
XGB Regression
Mean Squared Error of the model is  245845652159.13156
Mean Absolute Error of the model is  210884.81589999382
Root Mean Squared Error of the model is  495828.2486498037
-----------------------------------
```

We observe that the best Mean Absolute Error and Root Mean Squared Error are given by Extreme Gradient Boosting Algorithm. Following Gradient forest, Random Forest also displays a good performance on all metrics.

We plotted a graph to visualize the prediction.

Plotting the Graph to Visualize Prediction

```
plt.figure(figsize=(10,8))
plt.scatter(x=y_test,y=gbr1_pred,color='r')
plt.plot(y_test,y_test)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title("Gradient Boosting Regressor")
plt.show()
```



We Hypertuned the algorithm to check for any betterment.

Hypertuning

```
# Importing Random Search CV

from sklearn.model_selection import RandomizedSearchCV
```

```
gbr_param={'loss':['ls','lad','huber','quantile'],'learning_rate' :[0.001,0.01,0.1,0.2,0.3],'n_estimators':list(range(50,400,50))
rcv_gbr=RandomizedSearchCV(estimator=gbr,param_distributions=gbr_param,scoring='r2',cv=5)
```

```
rcv_gbr.fit(x_train,y_train)
rcv_gbr.best_params_
```

```
{'n_estimators': 150,
 'max_features': 'sqrt',
 'max_depth': 8,
 'loss': 'ls',
 'learning_rate': 0.2,
 'criterion': 'friedman_mse'}
```

```
# Cross validating GradientBoostingRegressor

gbr1 = GradientBoostingRegressor(n_estimators=150,max_features='sqrt',max_depth=8,loss='ls',learning_rate=0.2,criterion='friedmar
gbr1.fit(x_train, y_train)
gbr1_pred=gbr1.predict(x_test)
testing_accu=r2_score(y_test,gbr1_pred)*100
for k in range(2,10):
    cv_score=cross_val_score(gbr1,x,y,cv=k)
    cv_mean=cv_score.mean()*100
    print(f'At crossfold {k} the CV score is {cv_mean} and the accuracy for testing is {testing_accu} ')
    print('\n')

print('Mean Squared Error of the model is ',mean_squared_error(y_test,gbr1_pred))
print('Mean Absolute Error of the model is ',mean_absolute_error(y_test,gbr1_pred))
print('Root Mean Squared Error of the model is ',np.sqrt(mean_squared_error(y_test,gbr1_pred)))
```

```
At crossfold 2 the CV score is 55.7757436774573 and the accuracy for testing is 84.7605692094152


At crossfold 3 the CV score is 52.45664275175491 and the accuracy for testing is 84.7605692094152


At crossfold 4 the CV score is 62.79495693173942 and the accuracy for testing is 84.7605692094152


At crossfold 5 the CV score is 68.08027297282152 and the accuracy for testing is 84.7605692094152


At crossfold 6 the CV score is 63.867203772806555 and the accuracy for testing is 84.7605692094152


At crossfold 7 the CV score is 62.9115719728544 and the accuracy for testing is 84.7605692094152


At crossfold 8 the CV score is 63.958114933813626 and the accuracy for testing is 84.7605692094152


At crossfold 9 the CV score is 71.57060165162248 and the accuracy for testing is 84.7605692094152


Mean Squared Error of the model is  237742745549.05917
Mean Absolute Error of the model is  192681.5187157498
Root Mean Squared Error of the model is  487588.70531325805
```

After hyper-tuning the model although we didn't find an improvement in the R2 Score, the cross-validation score at CV=5 increases while Mean Absolute Error and RMSE certainly decreases. This we will move ahead with Extreme Gradient Boosting Regressor as our final model with tuned parameters to have better MAE and R2 score and RMSE.

Key Metrics for success in solving the problem under consideration

Metrics used for solving the current scenario:

R2 Score: The coefficient of determination, denoted R^2 or r^2 and pronounced "R squared", is the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It is a statistic used in the context of statistical models whose main purpose is either the prediction of future outcomes or the testing of hypotheses, based on other related information. It provides a measure of how well-observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

There are several definitions of R^2 that are only sometimes equivalent. One class of such cases includes that of simple linear regression where r^2 is used instead of R^2. When an intercept is included, then r^2 is simply the square of the sample correlation coefficient (i.e., r) between the observed outcomes and the observed predictor values. If additional regressors are included, R^2 is the square of the coefficient of multiple correlations. In both such cases, the coefficient of determination normally ranges from 0 to 1.

Mean Absolute Error: In the context of machine learning, absolute error refers to the magnitude of difference between the prediction of observation and the true value of that observation. MAE takes the average of absolute errors for a group of predictions and observations as a measurement of the magnitude of errors for the entire group. MAE can also be referred to as L1 loss function.

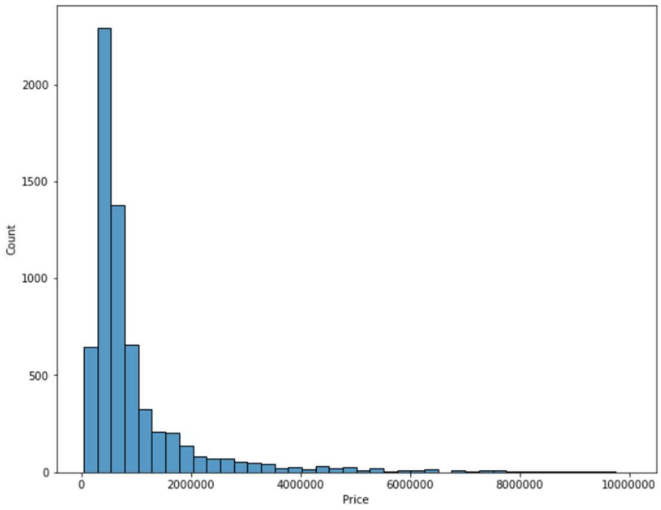
In our model, we get the mean absolute error as 192681.51 which is the least in all the models. This denotes the least difference between predicted and observed values in our dataset.

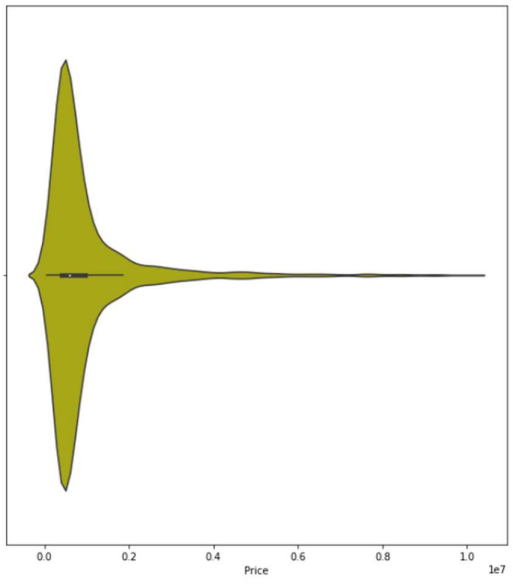
Root Mean Squared Error: Another popular way to calculate the error in a set of regression predictions is to use the Root Mean Squared Error. Shortened as RMSE, the metric is sometimes called Mean Squared Error or MSE, dropping the Root part from the calculation and the name. RMSE is calculated as the square root of the mean of the squared differences between actual outcomes and predictions. Squaring each error forces the values to be positive, and the square root of the mean squared error returns the error metric to the original units for comparison.

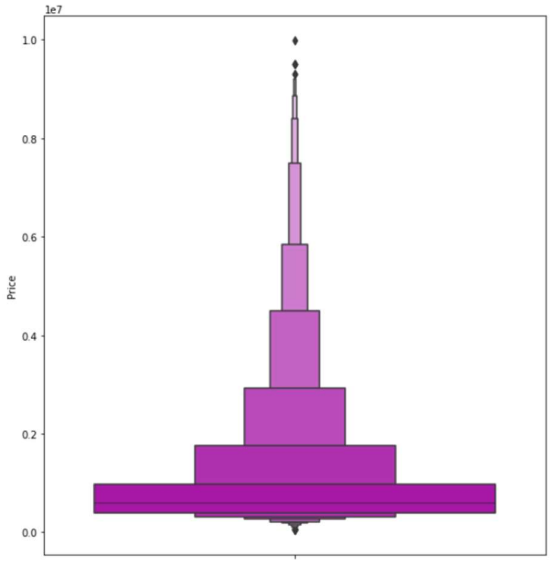
Visualizations

We performed analysis on the dataset and inferred some info regarding the pricing pattern and other features of used cars.

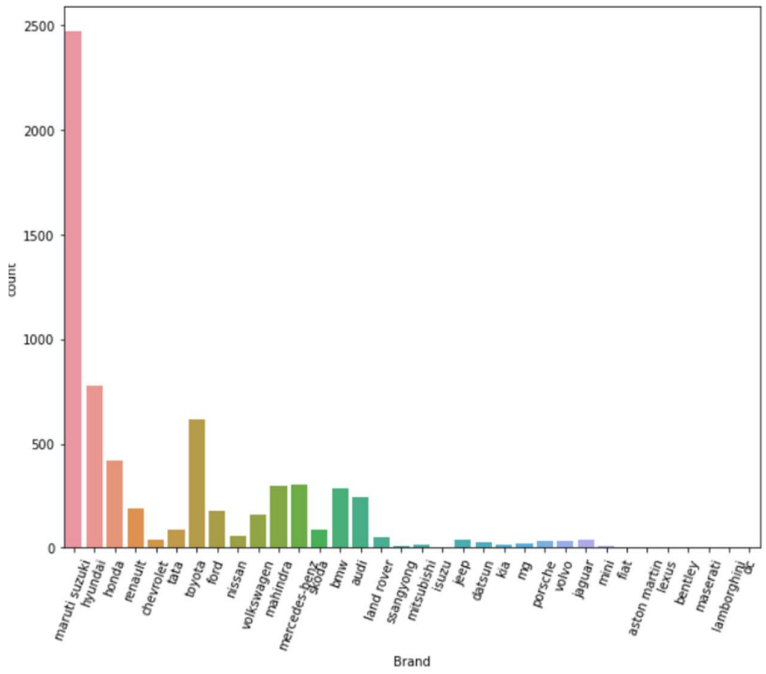
Visualizing and analyzing Target Variable.





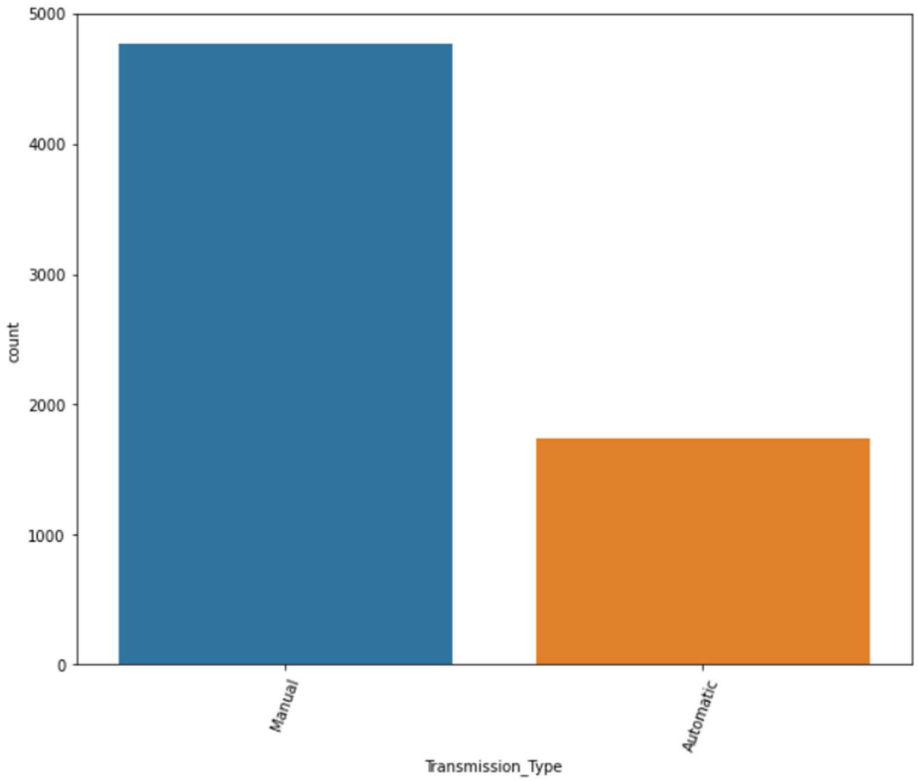


We see price majorly lies between 5 to 10 Lakhs. We can say that the Indian used car market is generally between this segment. We also see a significant number between 10 to 20 lakhs. While getting a used car in the range of 30 lakhs or above doesn't show any numbers.



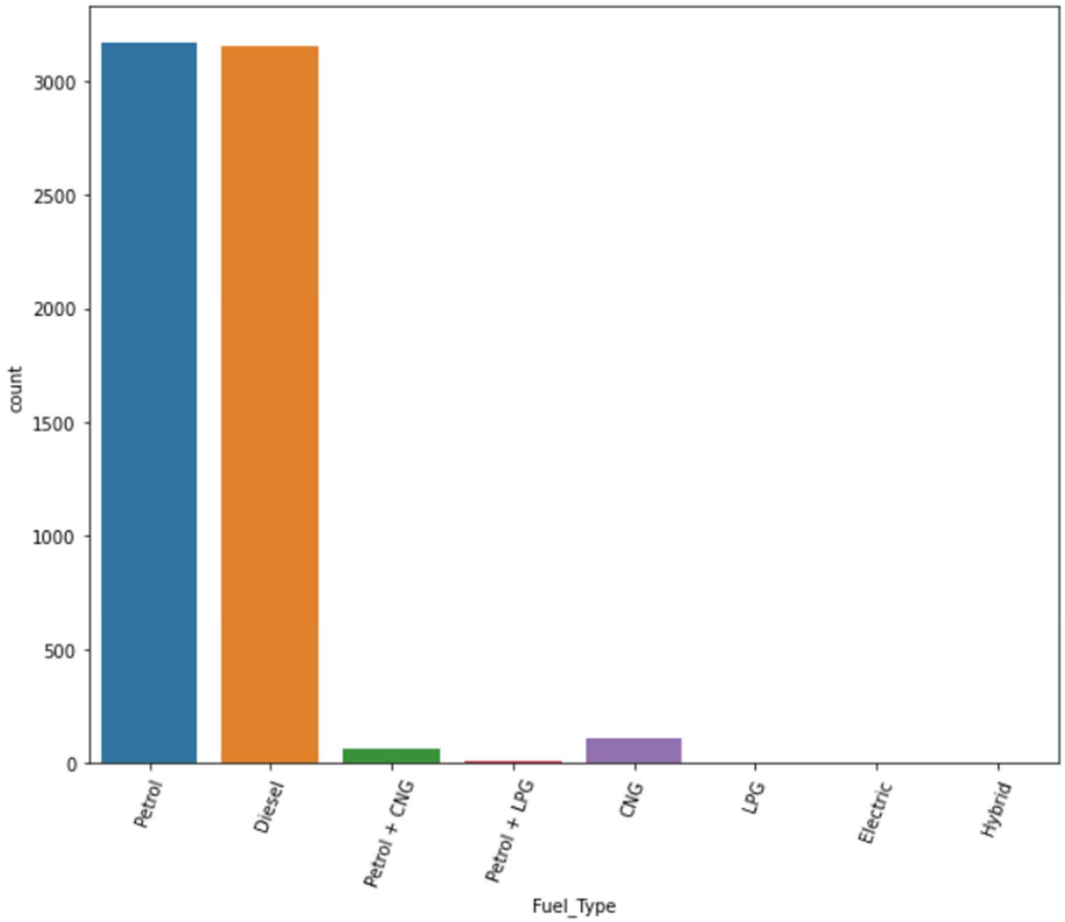
maruti suzuki	37.992005
hyundai	12.007995
toyota	9.471095
honda	6.472940
mercedes-benz	4.658672
mahindra	4.551046
bmw	4.381919
audi	3.705412
renault	2.890529
ford	2.706027
volkswagen	2.429274
tata	1.337638
skoda	1.291513
nissan	0.845633
land rover	0.799508
jeep	0.599631
jaguar	0.584256
chevrolet	0.568881
porsche	0.522755
volvo	0.492005
datsun	0.369004
mg	0.292128
kia	0.230627
mitsubishi	0.184502
ssangyong	0.169127
mini	0.153752
isuzu	0.061501
bentley	0.061501
aston martin	0.046125
fiat	0.030750
maserati	0.030750
lamborghini	0.030750
lexus	0.015375

Observation - We see the highest numbers for Maruti Suzuki i.e 37.99% followed by Hyundai at 12%. No doubt these Brands dominate the mid-range market. Also, this shows a high trend for resale for these brands. Other favs are Toyota and honda.

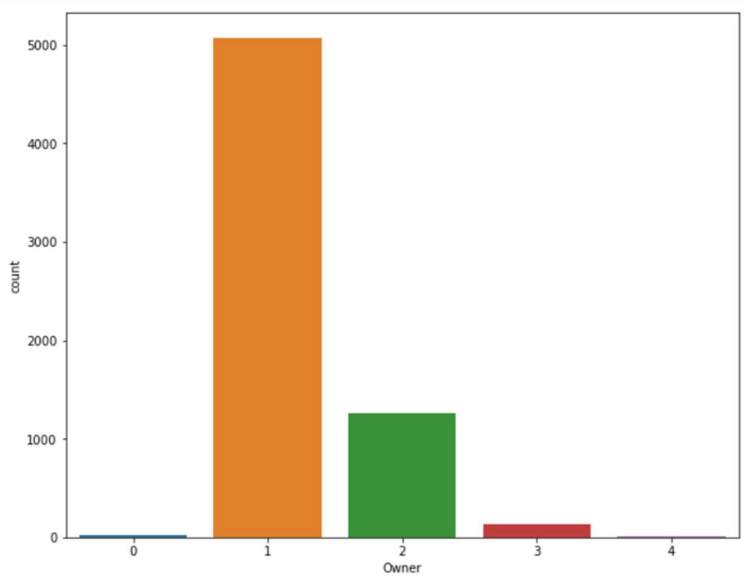



```
Manual       73.216482
Automatic    26.783518
Name: Transmission_Type, dtype: float64
```

73.21% vehicles have manual tranmission while 26.78% have automatic.

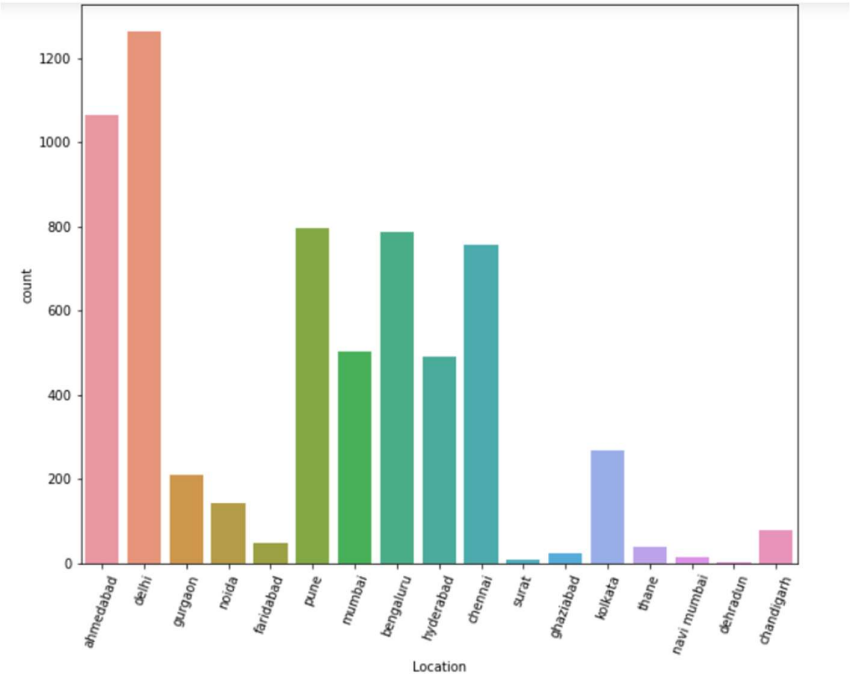


Both major fuels have an almost equal percentage. only 1.70% of cars have CNG. There are some vehicles using duel fuel type and also a Hybrid vehicle.



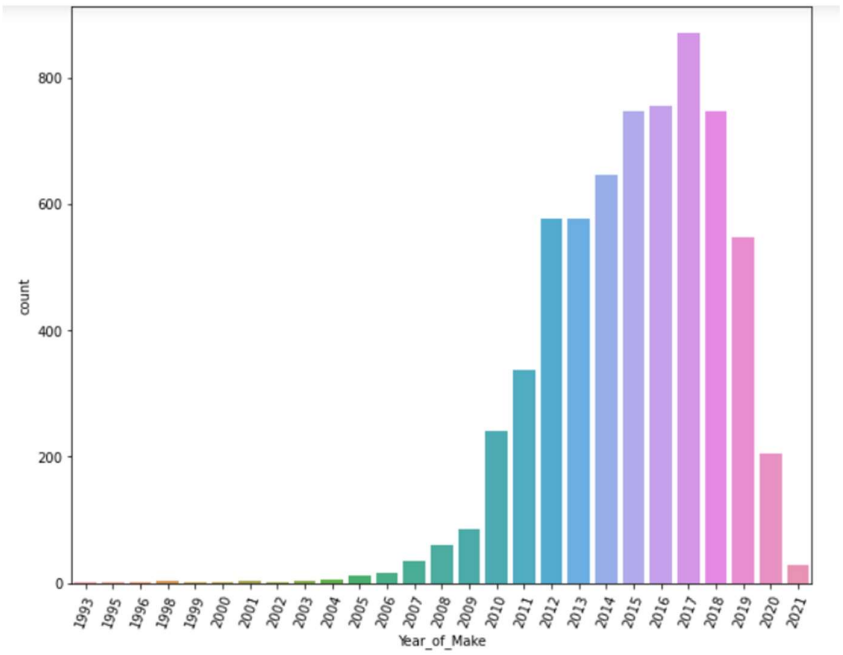

```
1    78.059656
2    19.449569
3     1.983395
0     0.307503
4     0.199877
Name: Owner, dtype: float64
```

The majority of cars are 1st owner i.e. 78.05%. People usually prefer 1st owned cars. 19.44% are 2nd owner owned cars.



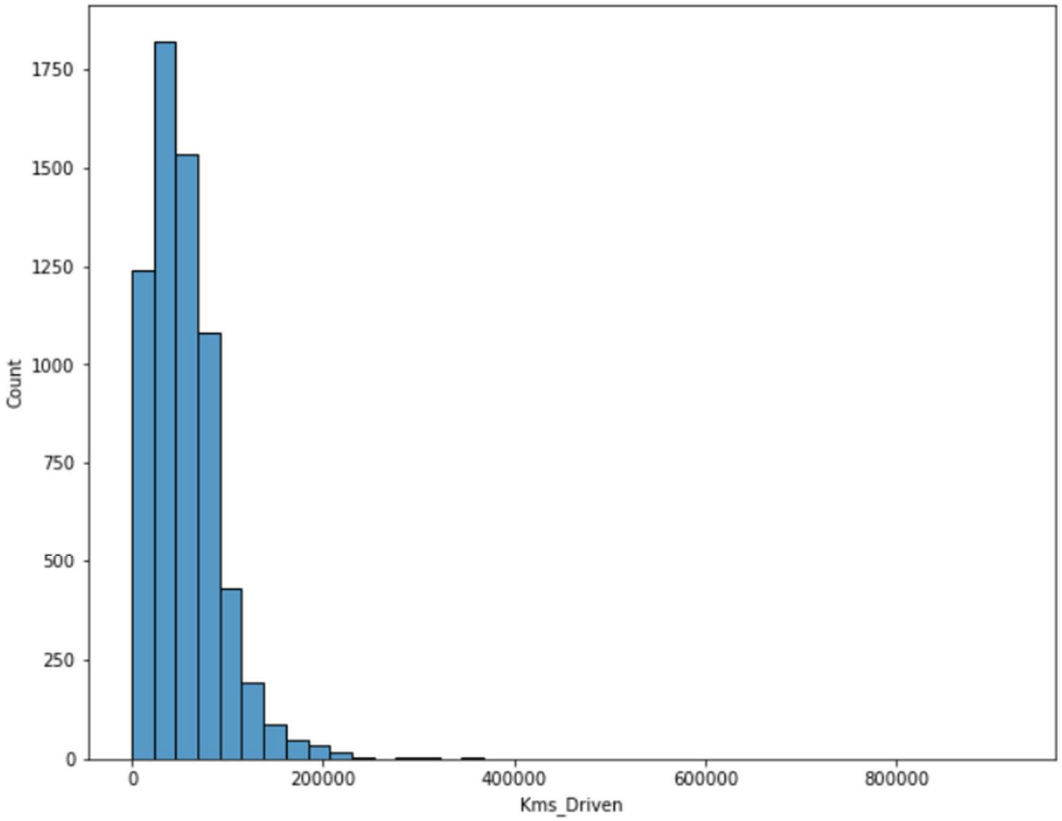

```
delhi          19.418819
ahmedabad      16.343788
pune           12.238622
bengaluru      12.100246
chennai        11.638991
mumbai          7.718327
```

19.41% cars are from delhi followed by ahmedabad, pune,bengaluru,chennai with 16.34%, 12.23%, 12.10%, 11.63% stacks.



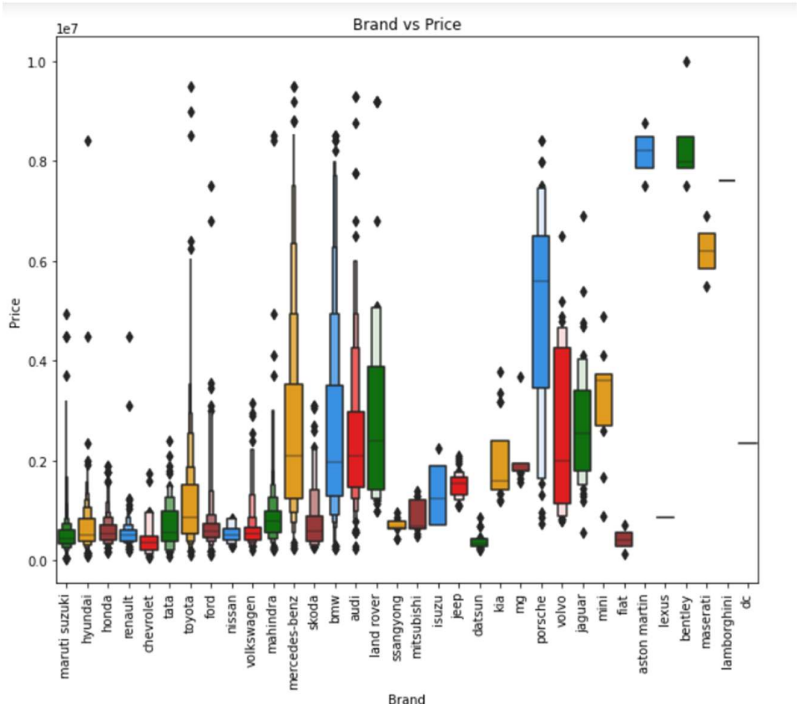

```
2017    13.376384
2016    11.608241
2015    11.485240
2018    11.469865
2014     9.916974
2013     8.871464
2012     8.856089
2019     8.410209
2011     5.196802
```

Stats show that the market is filled with cars that are 3-6 years old. This might mean that people prefer buying this old car.

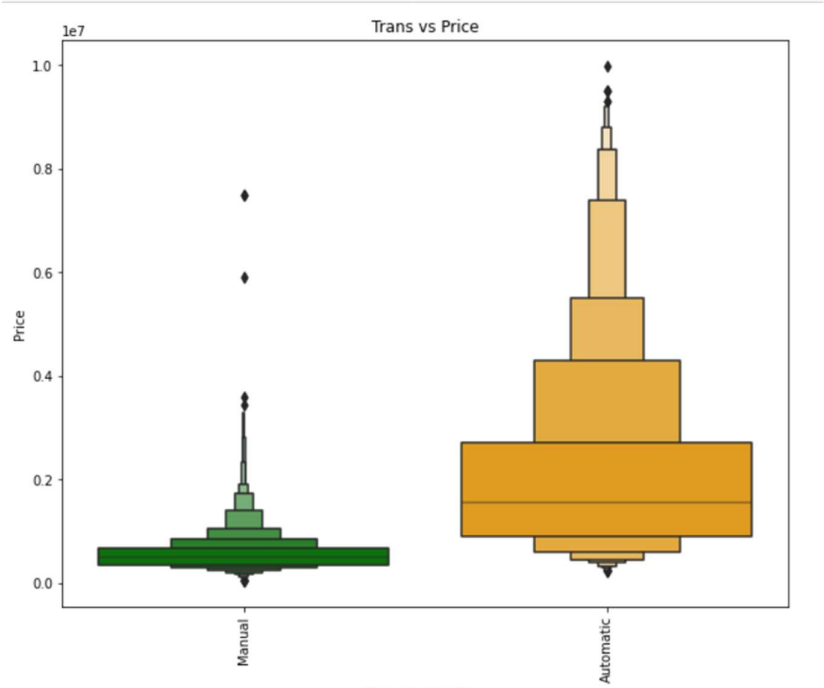


The majority of vehicles have 20k to 100k on the meter. We see significant numbers between 100k to 200k as well. Some entries show km higher then 200k as well.

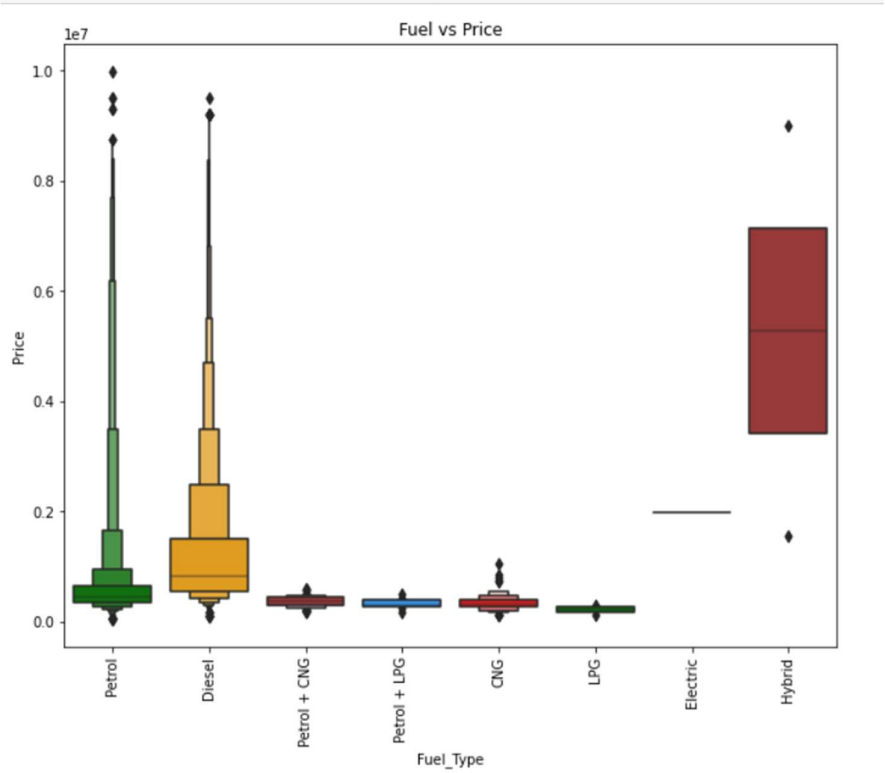
Checking Relation with Target Variable



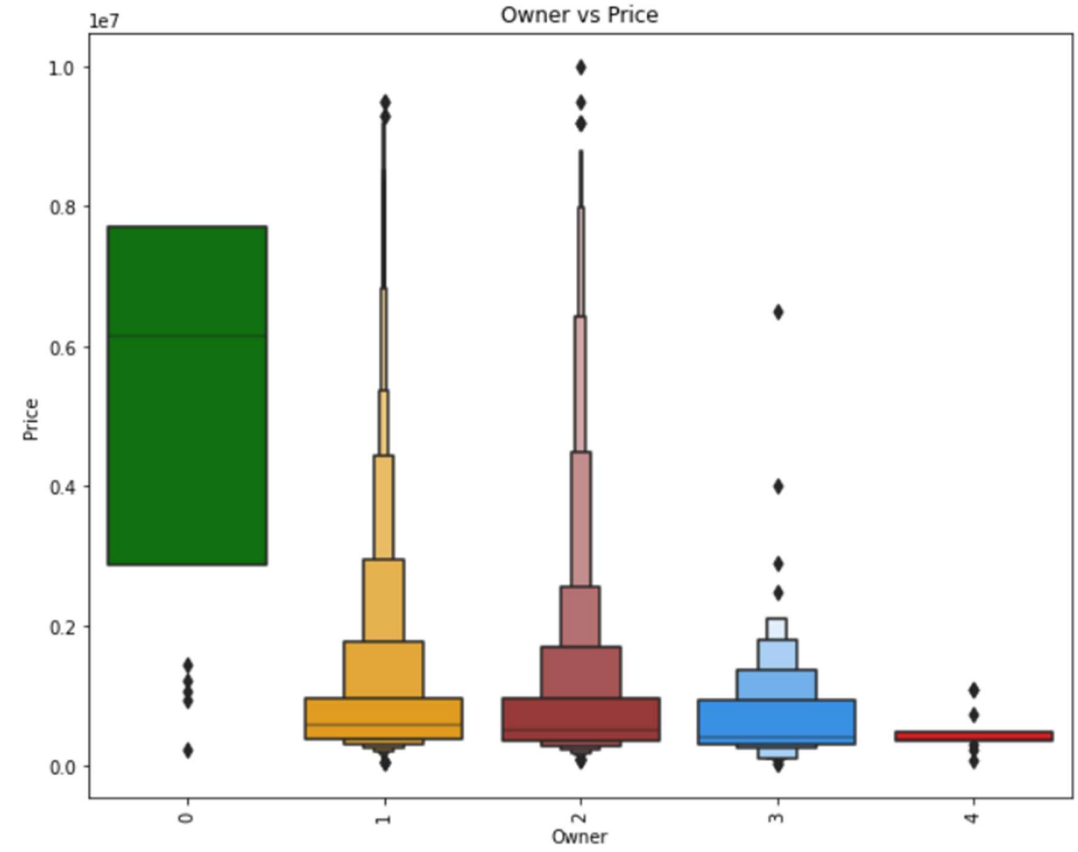
Maruti Suzuki brand which has the highest share stay on the lower price side. Some variants move up to 8000k. We see an extended range in Mercedes-Benz and BMW having low to high price range. Fiat Datsun marks the lower prices. Porsche though sits on a rather expensive section. Bentley Maserati has high prices.



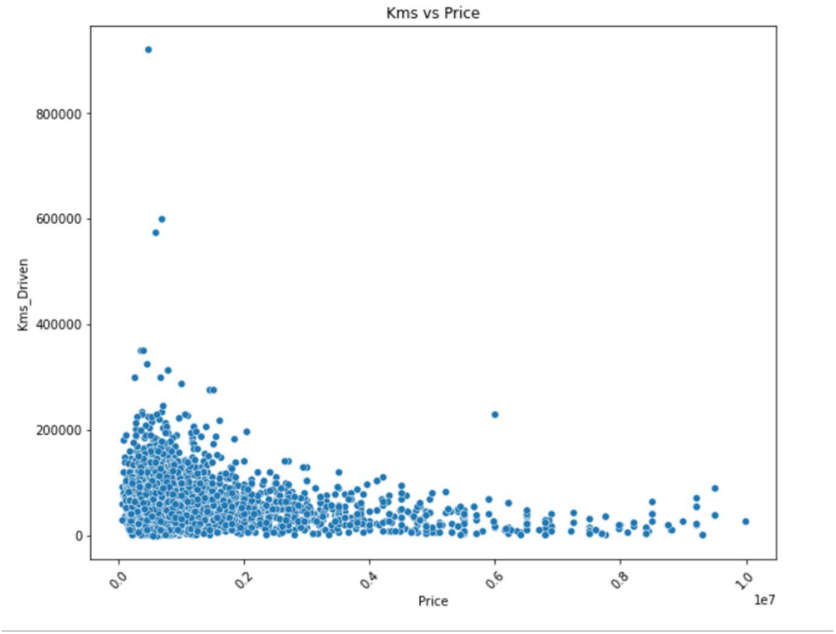
Although we saw higher counts for Manual trans, prices for automatics are on the higher side.



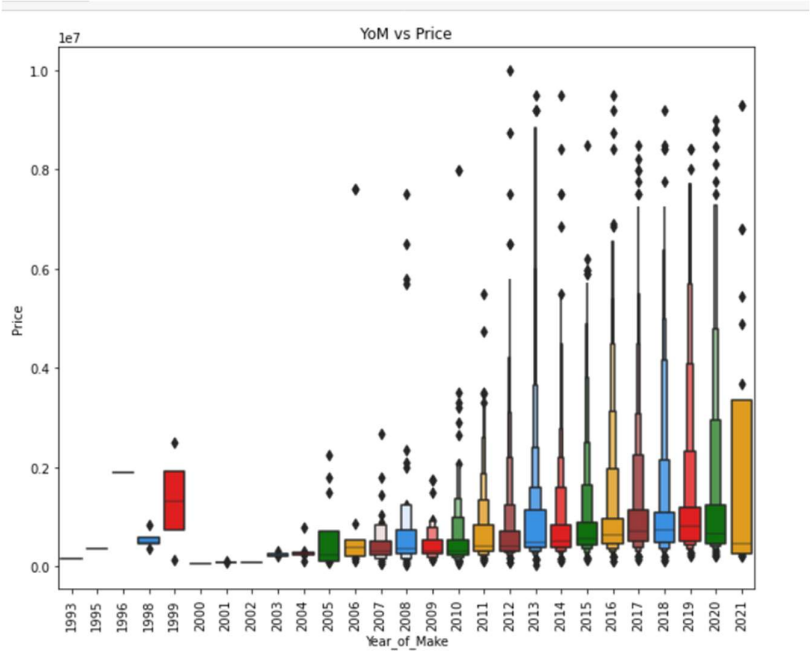
Both fuels move from low to high range price, however, we see diesel vehicles has extended range in the lower side. Petrol vehicles with low to mid-range usually have lower prices.



Not much price difference is seen when it comes to Owner count. Although owner count more than 2 hurts prices

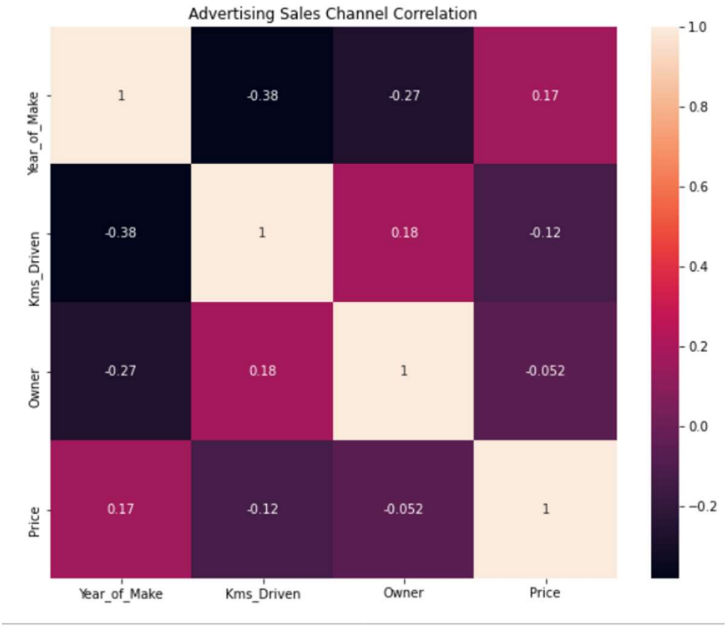


A negative relation is seen. Its obvious cuz the more the car has been driven, more impact it is going to make on its machinery thus, reducing the price.



Positive relation with the year of make.

Correlation between the features and with the target variable



We also devised a list of correlation values with the target variable.

```
: df2_cor['Price'].sort_values(ascending=False)

: Price          1.000000
  Year_of_Make   0.170212
  Owner         -0.052356
  Kms_Driven    -0.123522
  Name: Price, dtype: float64
```

.

Interpretation of the Results

We observe a positive relation of some features with our target variable. Certain brands have more resale potential while others have luxury and comfort. We observe that more cars are in the mid-range which extends from 4.5lacs to around 15-17 lacs. We also observe entry of luxury brands and preferences people show towards costly used cars have increased.

We observed that the features like low kms count, newer models, and adding on single-owner vehicles can fetch a good price. Also, location plays a crucial role has more impact on price.

CONCLUSION

Key Findings and Conclusions of the Study

The price of the car is based on various features all together. Starting from the brand itself to the KMS it has been driven has an impact on the price. Here also we tried to find the best combination to get the best prediction. A variant is somehow less effective as it has very less impact on the price of used cars.

Learning Outcomes of the Study in respect of Data Science

As they say, human beings are visual beings. Visualizations hold the power to convey the information most effectively and efficiently. Whether it be analytical visualization or graphical visualization, both play an important role in the analysis. Visualization is a great method to easily sight the information and various relation between the features. Using visualization also gives a quick glimpse of the situation.

With visualization, we can have an insight into the dataset. We can visualize data distribution, errors, and anomalies. Once the errors are found it is extremely important to treat the dataset to make it error-free and normalize the data. Data cleaning and correction is a must step before feeding the data into the machine learning algorithm. A clean and well-distributed data result in the machine learning model being more efficient and accurate.

In our case, we observe the best performance from the Extreme Gradient Boosting Algorithm which belongs to the boosting class of Ensemble Techniques. These algorithms often work on the principle of converting weak learners to strong learners. Boosting algorithms follow the approach of making weak rules and combine them to form strong rules. To find a weak rule, we apply base learning (ML) algorithms with a different distribution. Each time a base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Limitations of this work and Scope for Future Work

In the given dataset we observe that several features are available for us to train the model. Though major ones are covered still some could be included. With more samples populating minority objects model will have a better understanding. Here as we see the market is dynamic and prices can fluctuate frequently. Thus, a model trained at a particular point in time might not be effective enough.