FLIP ROBO

Car Price Prediction

Submitted by: Ojasav Sahu

ACKNOWLEDGMENT

First of all, I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity. The Data was collected from the “Cars24” website. Most of the concepts used to predict the price of the used cars are learned from Data Trained Institute. Here I would be thankful that I got this chance to do the project, this gave me good knowledge about the data collection and model building ie., prediction of the data.

INTRODUCTION

- With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper.

- The market value is based on a number of factors, including demand, supply, options, and incentives. The market value of a vehicle usually falls somewhere between the sticker price and the invoice price. Because the market value is an average, some people will pay more than that amount, while others will pay less.

- A car's value is determined by many factors: the popularity of the make and model of your car, vehicle specifications, trim levels, physical appearance, mileage, consistent maintenance and working condition. Using this as a base, I have collected the data from cars24 websites. The data was collected for the different car bodies.

- Once the data is collected, the data will be cleaned and pre-processed with all the necessary tools and the same will be used to build machine learning models in order to predict the price of the same.

Analytical Problem Framing

- The dataset has around 7039 rows and 14 columns. Using this dataset, we will be training the Machine Learning models on 70% of the data and the models will be tested on 30% data.

- Since we have removed the null values from the dataset during the data collection stage, we can expect outliers and un-realistic values for certain variables.

Importing the Required libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Data Collection :

```
data1 = pd.read_csv("cars1.csv")
data2 = pd.read_csv("cars2.csv")
data3 = pd.read_csv("cars3.csv")
data4 = pd.read_csv("cars4.csv")
data5 = pd.read_csv("cars5.csv")
data6 = pd.read_csv("cars6.csv")
data7 = pd.read_csv("cars7.csv")
data8 = pd.read_csv("cars8.csv")
data9 = pd.read_csv("cars9.csv")
data10 = pd.read_csv("cars10.csv")
data11 = pd.read_csv("cars11.csv")
data12 = pd.read_csv("cars12.csv")
data13 = pd.read_csv("cars13.csv")
```

Concatinating the data :

```
data = pd.concat([data1,data2,data3,data4,data5,data6,data7,data8,data9,data10,data11,data12,data13], axis = 0, ignore_index=True
data
```

0	Swift VDI ABS MANUAL	Maruti	['VDI', 'ABS']	MANUAL	Mar-15	76,264 km	76,264km (22 Nov 2021)	Diesel	1st Owner	Valid upto Mar 2023 3rd Party	Non-Accidental	DELHI	₹9,840/month	₹4,26,49
1	Swift ZDI MANUAL	Maruti	['ZDI']	MANUAL	Jul-14	92,088 km	92,088km (08 Mar 2022)	Diesel	1st Owner	Valid upto Mar 2023 3rd Party	Non-Accidental	DELHI	₹9,710/month	₹4,20,79
2	Baleno ALPHA DDIS 190 MANUAL	Maruti	['ALPHA', 'DDIS', '190']	MANUAL	Jan-16	67,332 km	67,332km (16 Nov 2021)	Diesel	1st Owner	Valid upto Jul 2022 Third Party	Non-Accidental	DELHI	₹13,612/month	₹5,92,19
3	Baleno ALPHA DDIS 190 MANUAL	Maruti	['ALPHA', 'DDIS', '190']	MANUAL	Nov-15	76,135 km	76,135km (07 Mar 2022)	Diesel	2nd Owner	Valid upto Mar 2023 3rd Party	Non-Accidental	DELHI	₹12,672/month	₹5,50,89
4	Baleno ZETA 1.2 K12 MANUAL	Maruti	['ZETA', '1.2', 'K12']	MANUAL	Mar-18	37,786 km	37,786km (09 Feb 2022)	Petrol	1st Owner	Valid upto Mar 2023 3rd Party	Non-Accidental	DELHI	₹14,546/month	₹6,33,19

Documentation : The data has lot of preprocessing to be done for attaining the better accuracy

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7039 entries, 0 to 7038
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Name                7039 non-null   object
 1   Brand               7039 non-null   object
 2   Model               7039 non-null   object
 3   Transmission        6956 non-null   object
 4   Year of Purchase    7039 non-null   object
 5   Kilometers Driven   7039 non-null   object
 6   Last Service        7039 non-null   object
 7   Fuel Type           7039 non-null   object
 8   Owner               7039 non-null   object
 9   Insurance           7039 non-null   object
 10  History             7039 non-null   object
 11  Location            6498 non-null   object
 12  EMI per month       7039 non-null   object
 13  Price               7039 non-null   object
dtypes: object(14)
memory usage: 770.0+ KB
```

Documentation : The columns of the data are completely objective datatype and also with few null-values in some

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

```
Name                   0
Brand                  0
Model                  0
Transmission          83
Year of Purchase       0
Kilometers Driven      0
Last Service           0
Fuel Type              0
Owner                  0
Insurance              0
History                0
Location             541
EMI per month          0
Price                  0
dtype: int64
```

The columns "Transmission" and "Location" have null values in them which have to be treated further

```
pd.set_option("display.max_columns",None)
data.describe()
```

	Name	Brand	Model	Transmission	Year of Purchase	Kilometers Driven	Last Service	Fuel Type	Owner	Insurance	History	Location	EMI per month	Price
count	7039	7039	7039	6956	7039	7039	7039	7039	7039	7039	7039	6498	7039	7039
unique	781	24	669	2	256	4371	4473	4	4	96	1	14	2906	3170
top	Baleno DELTA 1.2 K12 MANUAL	Maruti	['VXI']	MANUAL	Jan-14	36,696 km	1,12,006km (03 Jan 2022)	Petrol	1st Owner	Valid upto Mar 2023 3rd Party	Non-Accidental	DELHI	₹0/month	₹3,44,999
freq	241	3636	816	6024	167	7	7	4501	5610	5032	7039	1853	541	14

Documentation : The data here is not presented statistically as all the columns of the data are object datatype

- Now the **pre-processing** of the data is going to be done:
- Here the column **“Name”** is renamed by extracting the required data.

preprocess of the column "Name" :

```
data["Name"]
```

```
0                          Swift VDI ABS MANUAL
1                              Swift ZDI MANUAL
2                  Baleno ALPHA DDIS 190 MANUAL
3                  Baleno ALPHA DDIS 190 MANUAL
4                     Baleno ZETA 1.2 K12 MANUAL
                            ...
7034                        Alto 800 LXI MANUAL
7035      Elite i20 1.2 MAGNA PLUS VTVT MANUAL
7036                 maze 1.2 VXMT I VTEC MANUAL
7037         GRAND I10 NIOS MAGNA 1.2 MT MANUAL
7038           Grand i10 Sportz(O) 1.2 MT MANUAL
Name: Name, Length: 7039, dtype: object
```

```
x = data["Name"]

list0 = []

for i in x:
    list0.append(i.split(' ')[0])
list0
```

```
['Swift',
 'Swift',
 'Baleno',
 'Baleno',
 'Baleno',
 'Swift',
 'Swift',
 'Swift',
 'Ciaz',
 'Ciaz',
 'Baleno',
 'Ertiga',
 'Baleno',
 'Swift',
 'Ciaz',
 'Swift',
 'Swift',
 'Baleno',
 'Ertiga',
```

```
Name_of_the_car = list0

df = pd.DataFrame()
df["Name_of_the_car"] = list0
df
```

	Name_of_the_car
0	Swift
1	Swift
2	Baleno
3	Baleno
4	Baleno
...	...
7034	Alto
7035	
7036	maze
7037	
7038	

7039 rows × 1 columns

```
Car_name = df
```

```
data.insert(len(data.columns),"Car_name",Car_name.values)
```

Documentation : Here i have created a new column and inserted into the dataframe

```
data.drop(['Name'],axis = 1,inplace = True)
```

Documentation : Here i delete the main column "Name"

- Now, the column “**year of purchase**” is to be pre-processed.

Preprocess of the column " Year of Purchase " :

```
yea = data["Year of Purchase"]

list1 = []
for j in yea:
    list1.append(j.strip(' ')[0:3])
list1
```

```
 'Aug',
 'Jul',
 'Jan',
 'Sep',
 'Jan',
 'Jan',
 'Mar',
 'Jan',
 'Jun',
 'May',
 'Feb',
 'May',
 'May',
 'Aug',
 'Nov',
 'May',
 'Feb',
 'Mar',
 'Oct',
```

```
Purchase_month = list1

df1 = pd.DataFrame()
df1["Purchase_month"] = list1
df1
```

	Purchase_month
0	Mar
1	Jul
2	Jan
3	Nov
4	Mar
...	...
7034	Feb
7035	Jan
7036	Sep
7037	Aug
7038	Feb

7039 rows × 1 columns

```
Purchase_month = df1
```

```
list2 = []
for k in yea:
    list2.append(k.strip('')[-2:])
list2
```

```
 '17',
 '13',
 '15',
 '16',
 '16',
 '16',
 '16',
 '13',
 '15',
 '14',
 '14',
 '17',
 '14',
 '20',
 '17',
 '14',
 '14',
 '10',
 '14',
```

```
purchase_year = list2

df2 = pd.DataFrame()
df2["purchase_year"] = list2
df2
```

	purchase_year
0	15
1	14
2	16
3	15
4	18
...	...
7034	19
7035	19
7036	20
7037	17
7038	18

7039 rows × 1 columns

```
Purchase_year = df2
```

```
data.insert(len(data.columns),"Purchase_month",Purchase_month.values)
```

```
data.insert(len(data.columns),"Purchase_year",Purchase_year.values)
```

Documentation : The newly created columns are added to the dataframe

```
data.drop(['Year of Purchase'],axis = 1,inplace = True)
```

Documentation : Here i have deleted the main column "Year of Purchase".

- Here I have extracted 2 columns from the column “Year of purchase”, which are “**Purchase month**” and “**Purchase year**”.

Preprocess of the column "Kilometers Driven "

```
kms = data["Kilometers Driven"]

list3 = []
for l in kms:
    list3.append(l.split(" ")[0].replace(",",""))
list3
```

```
 '76530',
 '82367',
 '83937',
 '44193',
 '94820',
 '8235',
 '52121',
 '14734',
 '71382',
 '77787',
 '41996',
 '48259',
 '9610',
 '77409',
 '54884',
 '30528',
 '39011',
 '36696',
 '72310',
```

```
Kilometers_Driven = list3

df3 = pd.DataFrame()
df3["Kilometers_Driven"] = list3
df3
```

	Kilometers_Driven
0	76264
1	92088
2	67332
3	76135
4	37786
...	...
7034	16539
7035	31962
7036	15472
7037	6427
7038	35407

7039 rows × 1 columns

```
Kilometers_Driven = df3
```

```
data.insert(len(data.columns),"Kilometers_Driven",Kilometers_Driven.values)
```

Documentation : Here the new column created is inserted into the dataframe

```
data.drop(['Kilometers Driven'],axis = 1,inplace = True)
```

Documentation : The main column "Kilometers Driven" is dropped

- Here I have extracted the required information from the column "kilometres Driven" and instead created another column with same name " **kilometers_Driven"**.

Preprocess of the column "Last Service" :

```
service= data["Last Service"].str.split( '(' )

list4=[]
for m in range(len(service)):
    print(service[m][1][:-1])
    list4 += [service[m][1][:-1]]
list4
```

```
09 Feb 2022
17 Feb 2022
02 Feb 2022
27 Nov 2021
18 Feb 2022
14 Mar 2022
23 Feb 2022
11 Mar 2022
07 Feb 2022
08 Mar 2022
12 Mar 2022
20 Nov 2021
23 Feb 2022
07 Mar 2022
02 Feb 2022
17 Feb 2022
08 Mar 2022
24 Nov 2021
28 Feb 2022
24 Feb 2022
```

```
service = list4

df4 = pd.DataFrame()
df4["service"] = list4
df4
```

	service
0	22 Nov 2021
1	08 Mar 2022
2	16 Nov 2021
3	07 Mar 2022
4	09 Feb 2022
...	...
7034	24 Dec 2021
7035	01 Mar 2022
7036	21 Dec 2021
7037	21 Dec 2021
7038	11 Jan 2022

7039 rows × 1 columns

```
service = df4
```

```
data.insert(len(data.columns),"service",service.values)
```

Documentation : The new column created is added into the dataframe

```
data.drop(['Last Service'],axis = 1,inplace = True)
```

Documentation : Here i have dropped the main column "Last Service"

- Here I have dropped the main column "Last Service" and created another column **"service"** instead of it with required informatio

Preprocess of the column "Owner" :

```
own = data["Owner"]

list5 = []
for n in own:
    list5.append(n.strip(" ")[0])
list5
```

```
['1',
 '1',
 '1',
 '2',
 '1',
 '1',
 '1',
 '1',
 '2',
 '1',
 '2',
 '2',
 '1',
 '2',
 '1',
 '2',
 '1',
 '1',
 '2',
```

```
owner = list5

df4 = pd.DataFrame()
df4["owner"] = list5
df4
```

	owner
0	1
1	1
2	1
3	2
4	1
...	...
7034	2
7035	1
7036	1
7037	1
7038	1

7039 rows × 1 columns

```
owner = df4
```

```
data.insert(len(data.columns),"owner",owner.values)
```

Documentation : Here the newly created column is added into the dataframe

```
data.drop(['Owner'],axis = 1,inplace = True)
```

Documentation : Here i have dropped the main column "Owner"

- Here I have created a column with same name **"owner"** but with the required data and the dropped the main column "Owner".

Preprocess of the column "Insurance" :

```
ins = data["Insurance"]

list5 = []   ## insurance year
for p in ins:
    list5.append(p.split(" ")[2:4][1])
list5
```

```
['2023',
 '2023',
 '2022',
 '2023',
 '2023',
 '2023',
 '2023',
 '2023',
 '2023',
 '2023',
 '2023',
 '2023',
 '2022',
 '2023',
 '2022',
 '2023',
 '2023',
 '2023',
 '2023',
```

```
Year_of_insurance = list5

df5 = pd.DataFrame()
df5["Year_of_insurance"] = list5
df5
```

	Year_of_insurance
0	2023
1	2023
2	2022
3	2023
4	2023
...	...
7034	2023
7035	2023
7036	2023
7037	2023
7038	2022

7039 rows × 1 columns

```
Year_of_insurance = df5
```

```
data.insert(len(data.columns),"Year_of_insurance",Year_of_insurance.values)
```

Docmentation : Here the newly created column is added into the dataframe

```python
list6 = []
for q in ins:
    list6.append(q.split(" ")[2:4][0])
print(list6)
```

```
['Mar', 'Mar', 'Jul', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Sept', 'Mar', 'Sept', 'Mar', 'Mar', 'Ma
r', 'Mar', 'Mar', 'Aug', 'Mar', 'Mar', 'Mar', 'Mar', 'Sept', 'Mar', 'Mar', 'Mar', 'Jul', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar',
'Dec', 'Mar', 'Nov', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Nov', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar', 'Mar', 'Aug', 'Mar', 'Mar', 'Mar', 'Mar', 'Sept', 'Mar', 'Mar', 'Mar', 'Jul', 'Ma
r', 'Mar', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Nov', 'Mar', 'Mar', 'Sept', 'Nov', 'Mar', 'Mar', 'Nov', 'Aug', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Dec', 'Sep
t', 'Jul', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Nov', 'Mar', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'M
ar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Aug', 'Mar', 'Mar', 'Nov', 'Jan', 'Mar', 'Feb', 'Mar', 'Mar', 'Aug', 'Jul',
'Aug', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Jul', 'Mar', 'Mar', 'Mar', 'Mar', 'Nov', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Nov', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Nov',
'Mar', 'Sept', 'Mar', 'Mar', 'Jul', 'Mar', 'Jun', 'Mar', 'Jun', 'Mar', 'Mar', 'Jan', 'Mar', 'Mar', 'Mar', 'Mar', 'Jan', 'Fe
b', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Aug', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'M
ar', 'Mar', 'Mar', 'Mar', 'Nov', 'Mar', 'Aug', 'Mar', 'Nov', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Oct', 'Mar', 'Mar', 'Jul', 'Mar', 'Jan', 'Mar', 'Mar', 'Oct', 'Mar', 'Aug', 'Mar', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Jul', 'Mar', 'Jan', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Jul', 'Mar', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Oct', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Oct', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar',
'Mar', 'Mar', 'Mar', 'Nov', 'Aug', 'Nov', 'Jan', 'Mar', 'Mar', 'Mar', 'Dec', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar', 'Mar',
```

```python
Month_of_insurance = list6

df6 = pd.DataFrame()
df6["Month_of_insurance"] = list6
df6
```

	Month_of_insurance
0	Mar
1	Mar
2	Jul
3	Mar
4	Mar
...	...
7034	Mar
7035	Mar
7036	Mar
7037	Mar
7038	Oct

7039 rows × 1 columns

```python
Month_of_insurance = df6
```

```python
data.insert(len(data.columns),"Month_of_insurance",Month_of_insurance.values)
```

Documentation : Here the newly added column is added into the dataframe.

```python
data.drop(['Insurance'],axis = 1,inplace = True)
```

Documentation : Here the main column "Insurance" is deleted

- Here I have extracted 2 columns named **"Year of insurance"** and **"Month of insurance"** from the main column "insurance" and I have dropped that column.

Preprocess of the column "EMI per month" :

```
em = data["EMI per month"]

list8 = []   #emi
for s in em:
    list8.append(s.split("₹")[1].split("/")[0].replace(",",""))
list8
```

```
['9840',
 '9710',
 '13612',
 '12672',
 '14546',
 '7843',
 '10746',
 '7902',
 '11889',
 '13765',
 '13783',
 '16418',
 '11718',
 '8922',
 '13476',
 '8357',
 '8945',
 '13501',
 '11955',
```

```
EMI = list8

df8 = pd.DataFrame()
df8["EMI"] = list8
df8
```

	EMI
0	9840
1	9710
2	13612
3	12672
4	14546
...	...
7034	0
7035	0
7036	0
7037	0
7038	0

7039 rows × 1 columns

```
EMI = df8
```

```
data.insert(len(data.columns),"EMI",EMI.values)
```

Documentation : Here the newly created column is added into the dataframe

```
data.drop(['EMI per month'],axis = 1,inplace = True)
```

Documentation : Here i have deleted the main column "EMI per month".

- Here I have added a new column named **"EMI"** with data required from the main column "EMI per month" and then dropped that column.

Preprocess of the column "Service" :

```
servic = data["service"]

list9 = []
for t in servic:
    list9.append(t.split(" ")[1])
list9
```

```
['Nov',
 'Mar',
 'Nov',
 'Mar',
 'Feb',
 'Feb',
 'Feb',
 'Nov',
 'Feb',
 'Mar',
 'Feb',
 'Mar',
 'Feb',
 'Mar',
 'Mar',
 'Nov',
 'Feb',
 'Mar',
 'Feb',
```

```
service_month = list9

df9 = pd.DataFrame()
df9["service_month"] = list9
df9
```

	service_month
0	Nov
1	Mar
2	Nov
3	Mar
4	Feb
...	...
7034	Dec
7035	Mar
7036	Dec
7037	Dec
7038	Jan

7039 rows × 1 columns

```
service_month = df9
```

```
data.insert(len(data.columns),"service_month",service_month.values)
```

Documentation : Here the newly created column is added into the dataframe

```
list10 = []
for t in servic:
    list10.append(t.split(" ")[-1])
list10
```

```
 '2022',
 '2021',
 '2022',
 '2022',
 '2022',
 '2022',
 '2021',
 '2022',
 '2022',
 '2022',
 '2022',
 '2022',
 '2022',
 '2022',
 '2021',
 '2022',
 '2022',
 '2022',
 '2022',
 '2022',
```

```
service_year = list10

df10 = pd.DataFrame()
df10["service_year"] = list10
df10
```

	service_year
0	2021
1	2022
2	2021
3	2022
4	2022
...	...
7034	2021
7035	2022
7036	2021
7037	2021
7038	2022

7039 rows × 1 columns

```
service_year = df10
```

```
data.insert(len(data.columns),"service_year",service_year.values)
```

Documentation : Here the newly created column is added to the dataframe

```
data.drop(['service'],axis = 1,inplace = True)
```

Documenation : Here the main column "service" is dropped

- Here I have extracted two columns named **"Service year"** and **"Service month"** from the main column "Service" and then I have dropped the column.

Preprocess of the column "Price" :

```
pric = data["Price"]

list11 = []   #service m
for u in pric:
    list11.append(u.replace("₹","").replace(",",""))
list11
```

```
['426499',
 '420799',
 '592199',
 '550899',
 '633199',
 '338799',
 '466299',
 '341399',
 '516499',
 '598899',
 '599699',
 '715399',
 '508999',
 '386199',
 '586199',
 '361399',
 '387199',
 '587299',
 '519399',
```

```
Car_Price = list11

df11 = pd.DataFrame()
df11["Car_Price"] = list11
df11
```

	Car_Price
0	426499
1	420799
2	592199
3	550899
4	633199
...	...
7034	365599
7035	638999
7036	774999
7037	631399
7038	533099

7039 rows × 1 columns

```
Car_Price = df11
```

```
data.insert(len(data.columns),"Car_Price",Car_Price.values)
```

Documentation : Here the newly created column is added into the dataframe

```
data.drop(['Price'],axis = 1,inplace = True)
```

Documentation : Here i have dropped the main column "Price"

- Here I have extracted the required data from our label column “Price” and created a new column named **“Car_Price”** and I have dropped the main column.
- Here I am dropping the unnecessary column **“History”** in which all the values of the data are same and also do not have any impact on the data.

```
data["History"].unique()
```

```
array(['Non-Accidental'], dtype=object)
```

```
data.drop(['History'],axis = 1,inplace = True)
```

Documentation : Here the column "History" has single value in the entire column and is no addon connection with data and so i have dropped the column

Filling the null-values :

```
data['Location'] = data['Location'].fillna(data['Location'].mode()[0])
```

```
data['Transmission'] = data['Transmission'].fillna(data['Transmission'].mode()[0])
```

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

```
Brand                0
Model                0
Transmission         0
Fuel Type            0
Location             0
Car_name             0
Purchase_month       0
Purchase_year        0
Kilometers_Driven    0
owner                0
Year_of_insurance    0
Month_of_insurance   0
EMI                  0
service_month        0
service_year         0
Car_Price            0
dtype: int64
```

Documentation : Here there are no null values in the columns of the data

- Here I have filled the columns with null values and again I have checked the null values data which is “0” now in all the columns which means that I have successfully filled the null values in the columns necessary.

Changing the datatypes :

```
data['Purchase_year'] = data['Purchase_year'].astype('int')
```

```
data['Purchase_year'].dtype
```

```
dtype('int32')
```

```
data['Kilometers_Driven'] = data['Kilometers_Driven'].astype('int')
```

```
data['Kilometers_Driven'].dtype
```

```
dtype('int32')
```

```
data['owner'] = data['owner'].astype('int')
```

```
data['owner'].dtype
```

```
dtype('int32')
```

```
data['Year_of_insurance'] = data['Year_of_insurance'].astype('int')
```

```
data['Year_of_insurance'].dtype
```

```
dtype('int32')
```

```
data['EMI'] = data['EMI'].astype('int')
```

```
data['EMI'].dtype
```

```
dtype('int32')
```

```
data['service_year'] = data['service_year'].astype('int')
```

```
data['service_year'].dtype
```

```
dtype('int32')
```

```
data['Car_Price'] = data['Car_Price'].astype('int')
```

```
data['Car_Price'].dtype
```

```
dtype('int32')
```

Documentation : Here i have changed the datatypes of the created new columns into their required datatypes

- The newly created columns have the same datatype and so I have changed their datatypes as required for better model building.

Checking the datatypes again:

```
data.dtypes
```

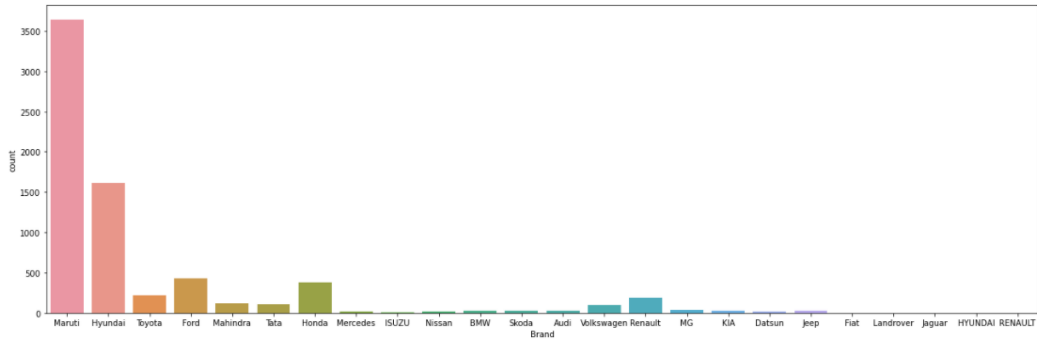
```
Brand                   object
Model                   object
Transmission            object
Fuel Type               object
Location                object
Car_name                object
Purchase_month          object
Purchase_year            int32
Kilometers_Driven        int32
owner                    int32
Year_of_insurance        int32
Month_of_insurance      object
EMI                      int32
service_month           object
service_year             int32
Car_Price                int32
dtype: object
```

Documenation : The new columns datatypes have been changed

❖ Here we start with the **Visualization** of the few features with label column and also their density plots:

Brand :

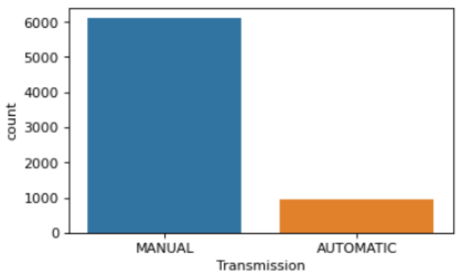
```
plt.figure(figsize=(22,7))
sns.countplot(data.Brand);
```



Documentation : Here the highest count is for "Maruti" and the least count is for "Nissan"

Transmission :

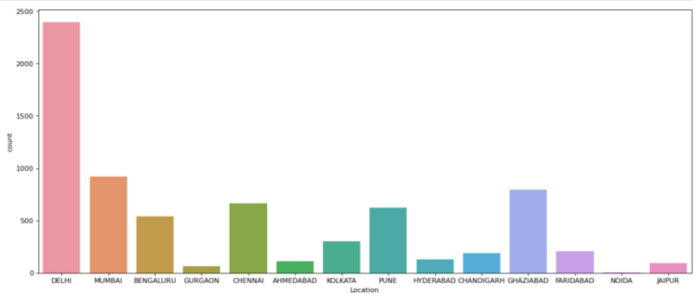
```
plt.figure(figsize=(5,3),dpi=80)
sns.countplot(data.Transmission);
```



Documentation : Here the highest count is for "Manual"

Location :

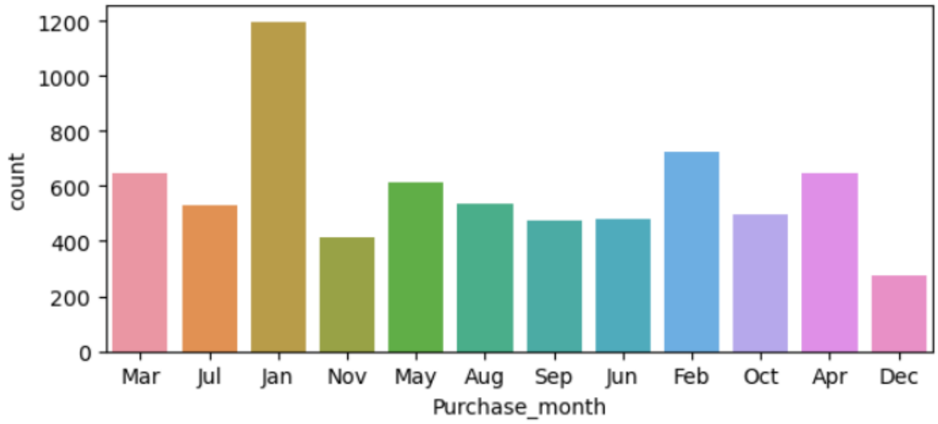
```
plt.figure(figsize=(17,7),dpi = 80)
sns.countplot(data.Location);
```



Documentation : Here the highest count is for "Delhi" and the least count is for "Noida"

Purchase_month :

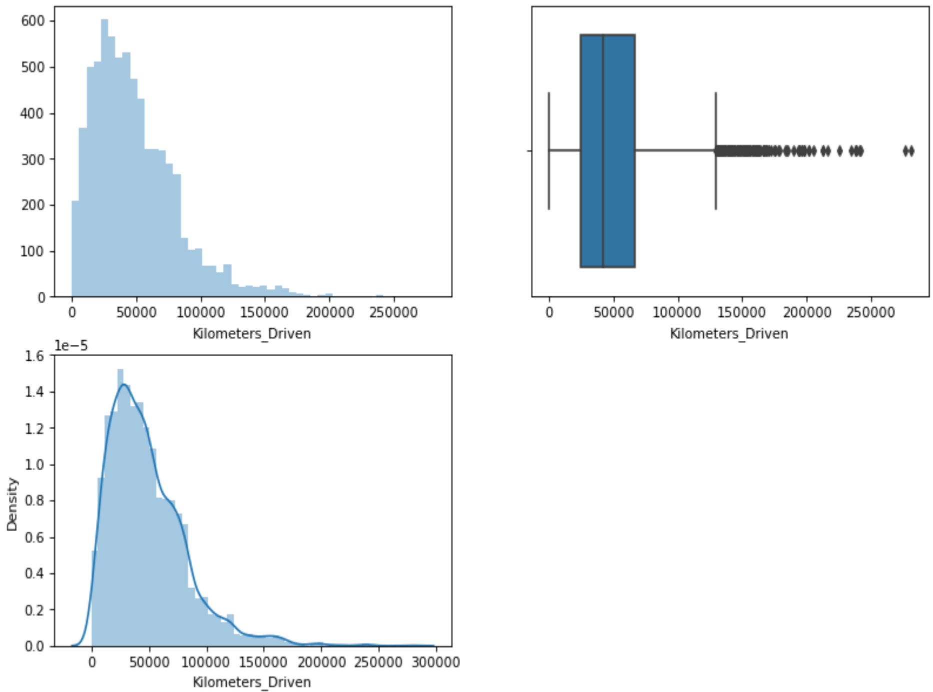
```
plt.figure(figsize=(7,3),dpi = 100)
sns.countplot(data.Purchase_month);
```



Documenation : Here the highest count is for "jan" and the least count is for "Dec"

Kilometers_Driven :

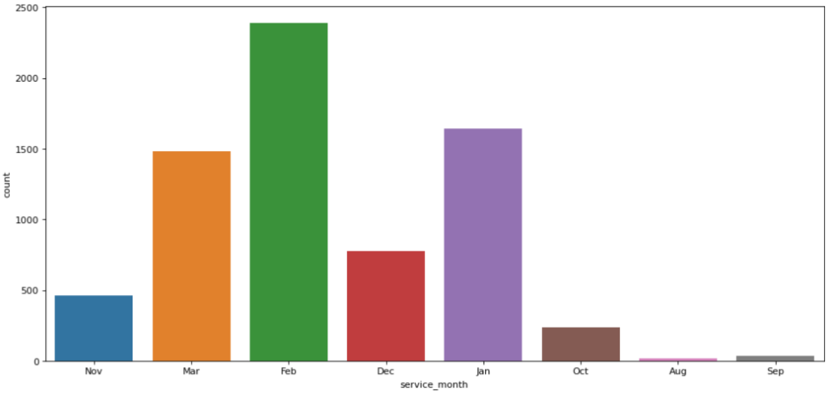
```
plt.figure(figsize=(12,8),dpi=80)
plt.subplot(2,2,1)
sns.distplot(data['Kilometers_Driven'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['Kilometers_Driven']);
plt.subplot(2,2,3)
sns.distplot(data['Kilometers_Driven']);
```



Documentation : Here the column has outliers and the distribution curve is skewed also

service_month :

```
plt.figure(figsize=(15,7),dpi = 80)
sns.countplot(data.service_month);
```



Documentation : Here the column has the highest count for "Feb" and the least count for "Aug"

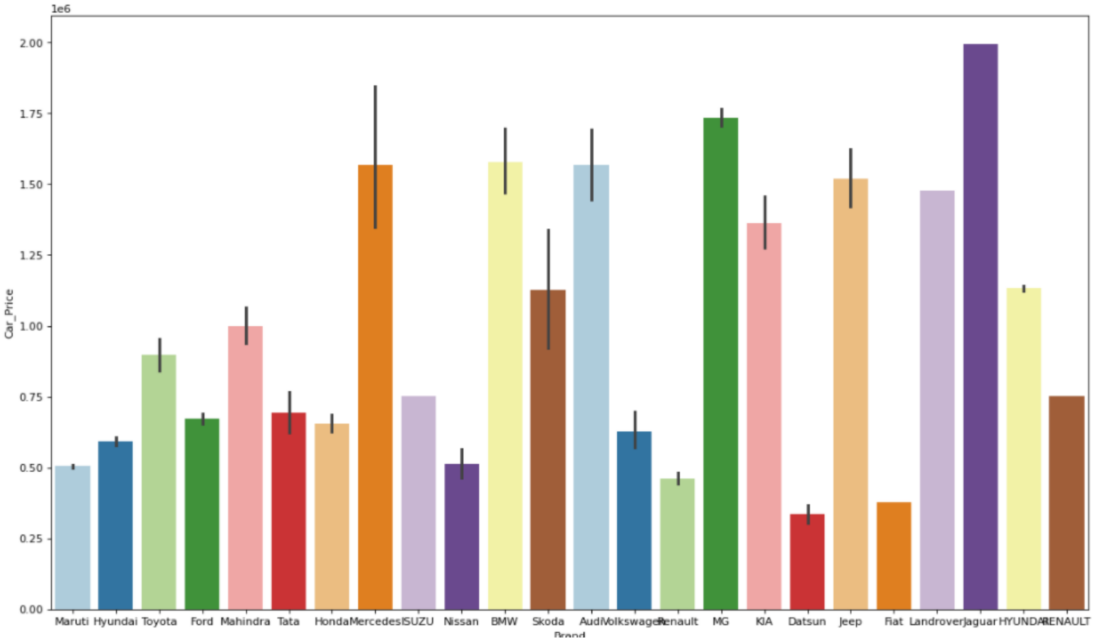
- Here I would be plotting the graph of the few features and the label, which is **Bivariate analysis :**

Bivariate :

Brand with Car_price :

```
plt.figure(figsize = (17,10),dpi = 80)
sns.barplot(x = 'Brand', y = 'Car_Price', data = data, palette = 'Paired')
```

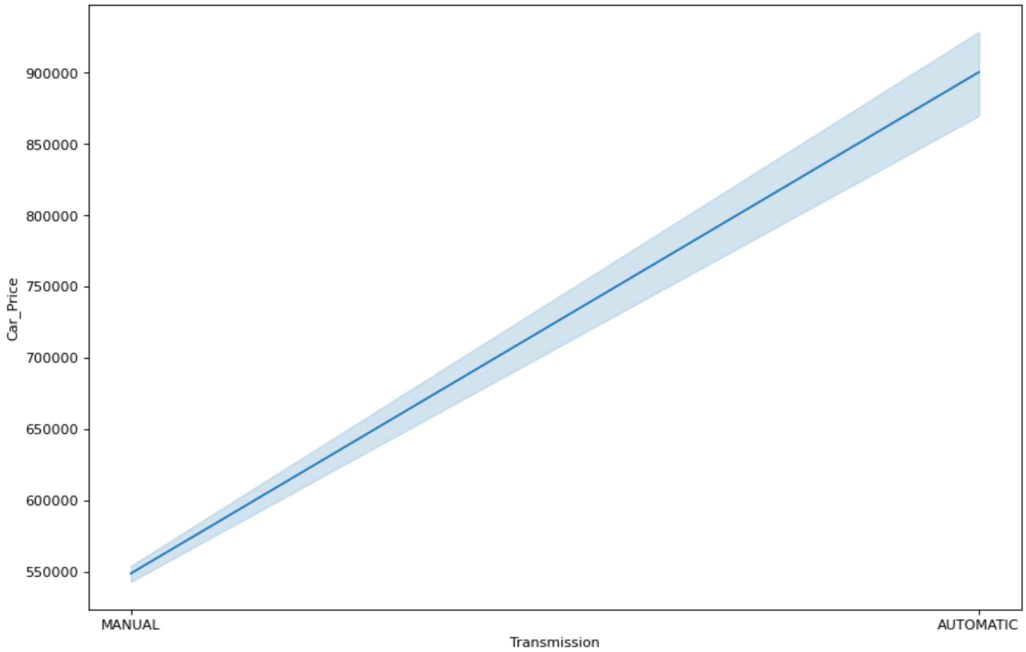
```
<AxesSubplot:xlabel='Brand', ylabel='Car_Price'>
```



Transmission with car_price :

```
plt.figure(figsize = (12,8),dpi = 80)
sns.lineplot(x = 'Transmission', y = 'Car_Price', data = data, palette = 'Cool')
```

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

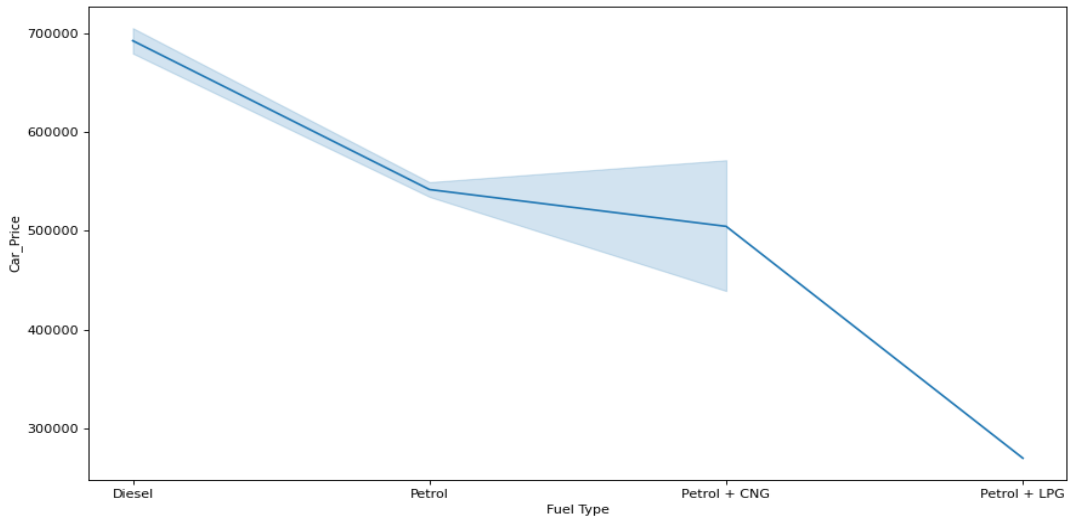


Documentation : Here the column has the linear relationship with the label

Fuel Type with car_price :

```
plt.figure(figsize = (13,7),dpi=80)
sns.lineplot(x = 'Fuel Type', y = 'Car_Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Fuel Type', ylabel='Car_Price'>
```

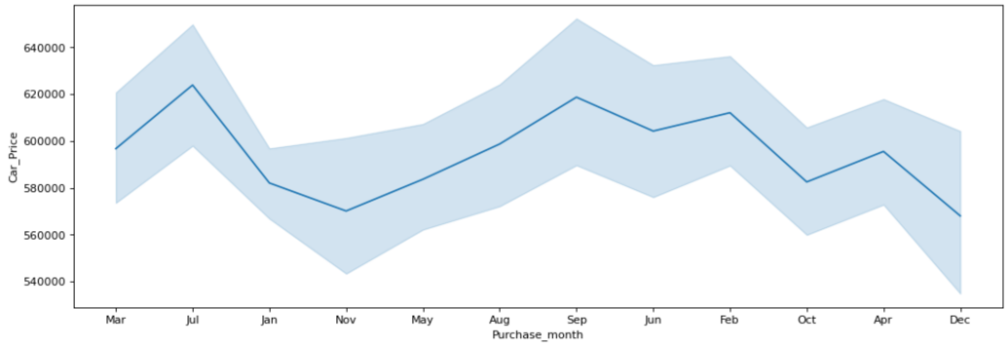


Documentation : Here the highest price is for Diesel cars than the other types of fuels

Purchase_month with car_price :

```
plt.figure(figsize = (15,5),dpi=80)
sns.lineplot(x = 'Purchase_month', y = 'Car_Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Purchase_month', ylabel='Car_Price'>
```

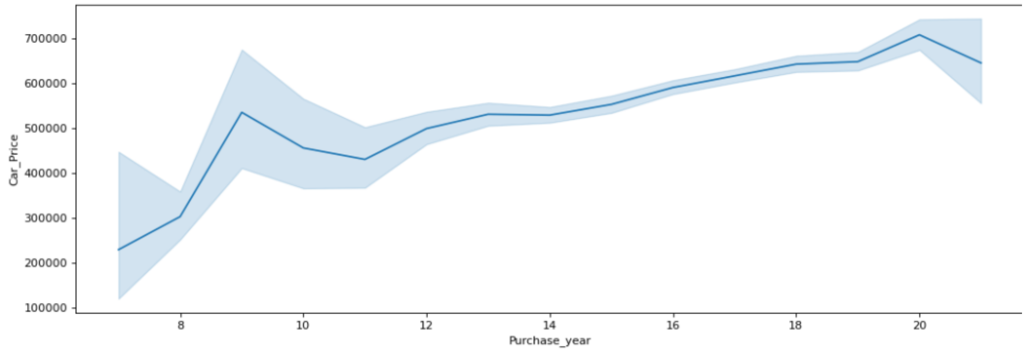


Documentation : Here the column has the high prices of the cars in different months in uneven ranges.

Purchase_year with Car_Price :

```
plt.figure(figsize = (15,5),dpi=80)
sns.lineplot(x = 'Purchase_year', y = 'Car_Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Purchase_year', ylabel='Car_Price'>
```

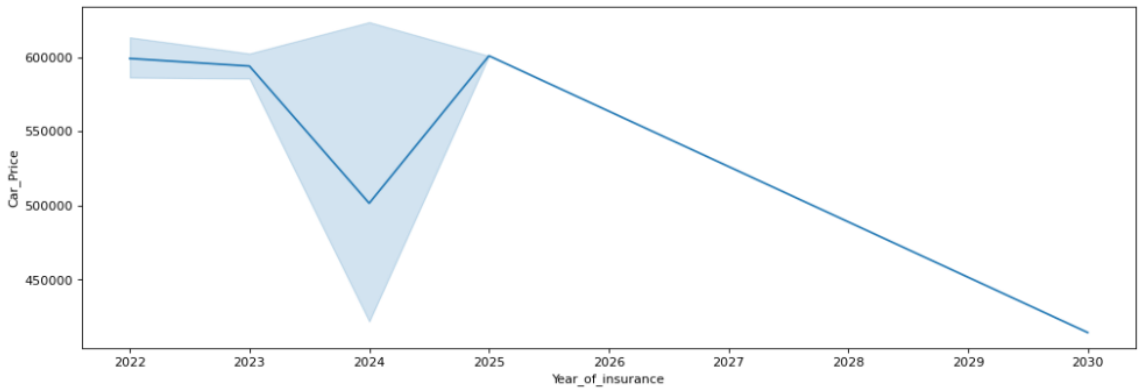


Documentation : Here the column has the highest price in the year "2020" than the other years

Year_of_insurance with Car_Price :

```
plt.figure(figsize = (15,5),dpi=80)
sns.lineplot(x = 'Year_of_insurance', y = 'Car_Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Year_of_insurance', ylabel='Car_Price'>
```

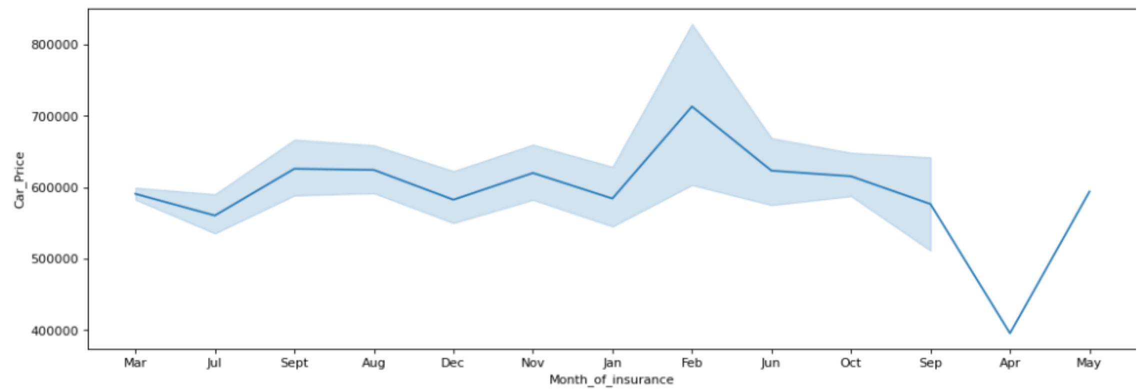


Documentation : Here the column has the highest price in the years between 2022 - 2025 at 600000

Month_of_insurance with Car_Price :

```
plt.figure(figsize = (15,5),dpi=80)
sns.lineplot(x = 'Month_of_insurance', y = 'Car_Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Month_of_insurance', ylabel='Car_Price'>
```



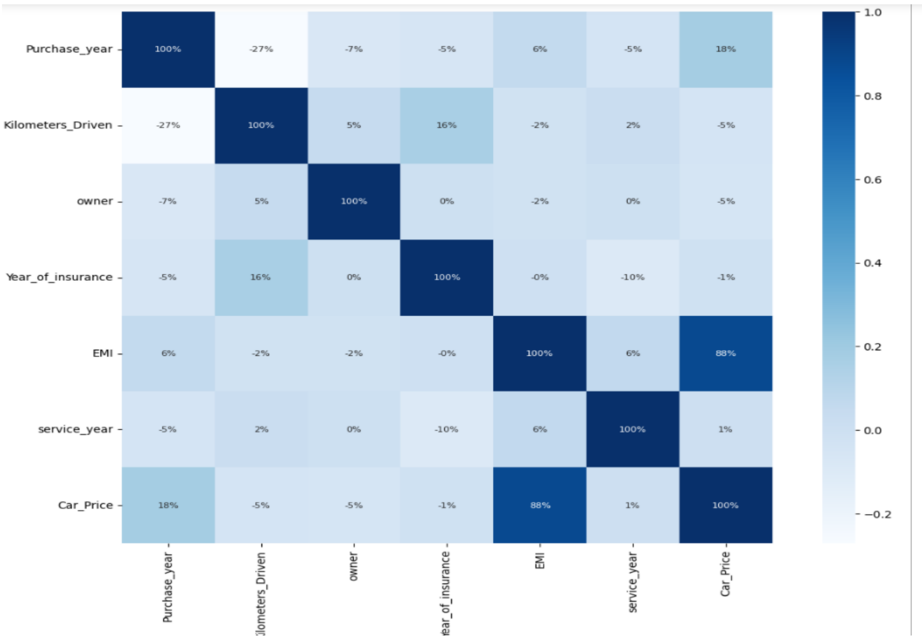
Documentation : Here the column has the uneven price range between march to july and is least in April and is somewhat high in Feb

- These are the few features for which univariate and bivariate analysis is shown and now we will be moving to the correlation part of the model.
- **Correlation** of the features with the label:

Correlation :

```
corr = data.corr()
```

```
plt.figure(figsize = (15,10),dpi=100)
sns.heatmap(corr,annot = True,fmt = ".0%",cbar = True,square = True,annot_kws = {'size':8}, cmap = 'Blues')
plt.show()
```



❖ Here the highest correlated column is the column **"EMI"** for our label column **"Car_Price".**

❖ Now, here we **encode the data** present in our dataset for further model building through **Label Encoder:**

```
from sklearn.preprocessing import LabelEncoder
```

```
for column in data.columns:
    if data[column].dtype == np.number:
        continue
    data[column] = LabelEncoder().fit_transform(data[column])
```

```
data
```

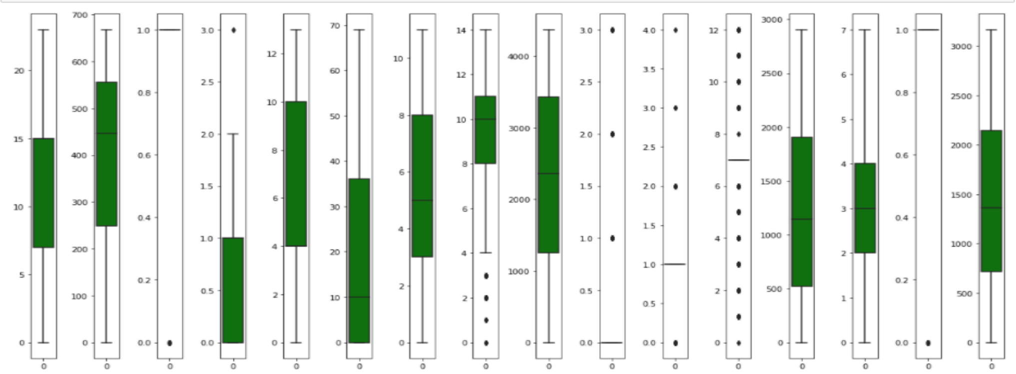
	Brand	Model	Transmission	Fuel Type	Location	Car_name	Purchase_month	Purchase_year	Kilometers_Driven	owner	Year_of_insurance	Month_of_insurar
0	15	528	1	0	4	32	7	8	3712	0	1	
1	15	626	1	0	4	32	5	7	4006	0	1	
2	15	227	1	0	4	4	4	9	3463	0	0	
3	15	227	1	0	4	4	9	8	3706	1	1	
4	15	633	1	1	4	4	7	11	2122	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	
7034	15	372	1	1	4	3	3	12	747	1	1	
7035	7	33	1	1	4	0	4	12	1768	0	1	
7036	6	65	1	1	4	59	11	13	683	0	1	
7037	7	380	1	1	4	0	1	10	195	0	1	
7038	7	493	1	1	4	0	3	11	1995	0	0	

7039 rows × 16 columns

Documentation : Here i have encoded the complete data for proceeding with our model

❖ Now, we will **check for the outliers** present in the data:

```
col_list = data.columns.values
ncol = 40
nrows = 20
plt.figure(figsize = (ncol,3*ncol))
for i in range (0, len(col_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(data = data[col_list[i]],color = 'green', orient = 'v')
    plt.tight_layout()
```



Documenation : Here there are few columns which have outliers in them which are to be treated

❖ We will be **Removing the outliers through Z-Score method:**

Removing the Outliers :

```
from scipy.stats import zscore
z = np.abs(zscore(data))
z.shape
```

```
(7039, 16)
```

```
threshold = 3.5
print(np.where(z>3.5))
```

```
(array([  47,   80,   83,  178,  215,  370,  550,  550,  622,  623,  646,
        735,  788,  854,  855,  862,  905,  962, 1001, 1016, 1091, 1116,
       1125, 1135, 1210, 1212, 1300, 1319, 1324, 1420, 1430, 1765, 1795,
       1797, 1839, 1889, 1901, 1902, 1905, 1937, 1944, 1945, 1962, 1965,
       2008, 2055, 2079, 2088, 2122, 2142, 2159, 2159, 2230, 2304, 2309,
       2641, 2716, 2825, 2832, 2865, 2899, 2948, 2961, 2971, 3012, 3016,
       3023, 3063, 3080, 3095, 3177, 3311, 3318, 3333, 3355, 3363, 3406,
       3449, 3471, 3508, 3519, 3570, 3584, 3648, 4049, 4192, 4215, 4219,
       4261, 4269, 4283, 4325, 4341, 4361, 4409, 4428, 4451, 4465, 4481,
       4488, 4498, 4542, 4561, 4562, 4605, 4657, 4692, 4727, 4735, 4786,
       4914, 4927, 4964, 4982, 4986, 4992, 5085, 5107, 5130, 5170, 5213,
       5253, 5278, 5287, 5297, 5360, 5382, 5383, 5479, 5499, 5504, 5609,
       5619, 5823, 5929, 5961, 6004, 6065, 6074, 6084, 6143, 6164, 6165,
       6253, 6271, 6276, 6377, 6386, 6583, 6702, 6742, 6824, 6962, 7028],
      dtype=int64), array([ 9,  9,  9,  9,  9,  9,  7,  9,  7,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        7,  9,  7,  9,  9,  9,  9,  7,  9,  9,  9,  9,  9,  9,  9,  9,  7,
        9, 10,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9, 10,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
       10,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  3,  9,
        9], dtype=int64))
```

```
new_data = data[(z<3.5).all(axis = 1)]
print(data.shape)
print(new_data.shape)
```

```
(7039, 16)
(6887, 16)
```

Documentation : Here i can see that the data has reduced because the number of records got reduced that means the outliers are successfully treated to some extent

❖ Now we will check the **Data Loss %** in our data:

Data loss :

```
data_loss = (7039-6823)/7039*100
```

```
data_loss
```

```
3.0686177013780367
```

Documentation : Here the loss% is less and is negligible

❖ Here I have data loss of 3.06% which is negligible and I can continue with the model building.

Now, here I am going to check the **skewness of the data:**

Checking the Skewness :

```
new_data.skew().sort_values()
```

```
Transmission          -2.197651
service_year          -1.368730
Year_of_insurance     -1.125785
Month_of_insurance    -0.728866
Model                 -0.516767
Fuel Type             -0.490141
Purchase_year         -0.347282
Brand                 -0.211308
Kilometers_Driven     -0.106630
Purchase_month         0.111530
Car_Price              0.219806
EMI                    0.229220
service_month          0.570786
Location               0.627537
Car_name               0.688809
owner                  1.614283
dtype: float64
```

- ❖ Here the number of columns with **negative skewness is more** than positive skewness columns and which have to be treated.

- ❖ Now, here we will **remove the skewness of the data** through **"PowerTransformer":**

Removing the skewness :

```
columns = ["Month_of_insurance","Transmission","service_year","Year_of_insurance","Location","owner","Car_name"]
```

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box-cox' or 'yeo-johnson'
'''
```

```
"\nparameters:\nmethod = 'box-cox' or 'yeo-johnson'\n"
```

```
new_data[columns] = scaler.fit_transform(new_data[columns].values)
new_data[columns]
```

	Month_of_insurance	Transmission	service_year	Year_of_insurance	Location	owner	Car_name
0	0.060945	0.386975	-1.895889	0.525504	-0.398386	-0.478022	0.830679
1	0.060945	0.386975	0.527457	0.525504	-0.398386	-0.478022	0.830679
2	-0.929616	0.386975	-1.895889	-1.662698	-0.398386	-0.478022	-0.475183
3	0.060945	0.386975	0.527457	0.525504	-0.398386	2.091953	-0.475183
4	0.060945	0.386975	0.527457	0.525504	-0.398386	-0.478022	-0.475183
...	...	...	...	...	...	...	...
7034	0.060945	0.386975	-1.895889	0.525504	-0.398386	2.091953	-0.606943
7035	0.060945	0.386975	0.527457	0.525504	-0.398386	-0.478022	-1.335193
7036	0.060945	0.386975	-1.895889	0.525504	-0.398386	-0.478022	1.326082
7037	0.060945	0.386975	-1.895889	0.525504	-0.398386	-0.478022	-1.335193
7038	1.794873	0.386975	0.527457	-1.662698	-0.398386	-0.478022	-1.335193

6887 rows × 7 columns

❖ Here, I have assigned few columns which have skewness to a variable and passed it through the **"Scaler.fit_transform"** method to reduce the skewness of the data.

```
new_data.skew()
```

```
Brand                -0.211308
Model                -0.516767
Transmission         -2.197651
Fuel Type            -0.490141
Location             -0.025472
Car_name             -0.119098
Purchase_month        0.111530
Purchase_year        -0.347282
Kilometers_Driven    -0.106630
owner                 1.614283
Year_of_insurance    -0.125557
Month_of_insurance    0.242302
EMI                   0.229220
service_month         0.570786
service_year         -1.368730
Car_Price             0.219806
dtype: float64
```

Documentation : Here the skewness values are changed somewhat to some extent

❖ Now, here I am going separate the dependent and independent variables x and y in which the all the features are assigned to variable "x" and label column is assigned to variable" y".

Separating the variables into independent and target variables :

```
x = new_data.drop("Car_Price", axis=1)
y = new_data["Car_Price"]
```

```
x
```

	Brand	Model	Transmission	Fuel Type	Location	Car_name	Purchase_month	Purchase_year	Kilometers_Driven	owner	Year_of_insurance	Month_of_ins
0	15	528	0.386975	0	-0.398386	0.830679	7	8	3712	-0.478022	0.525504	0.
1	15	626	0.386975	0	-0.398386	0.830679	5	7	4006	-0.478022	0.525504	0.
2	15	227	0.386975	0	-0.398386	-0.475183	4	9	3463	-0.478022	-1.662698	-0.
3	15	227	0.386975	0	-0.398386	-0.475183	9	8	3706	2.091953	0.525504	0.
4	15	633	0.386975	1	-0.398386	-0.475183	7	11	2122	-0.478022	0.525504	0.
...	...	...	...	...	...	...	...	...	...	...	...	
7034	15	372	0.386975	1	-0.398386	-0.606943	3	12	747	2.091953	0.525504	0.
7035	7	33	0.386975	1	-0.398386	-1.335193	4	12	1768	-0.478022	0.525504	0.
7036	6	65	0.386975	1	-0.398386	1.326082	11	13	683	-0.478022	0.525504	0.
7037	7	380	0.386975	1	-0.398386	-1.335193	1	10	195	-0.478022	0.525504	0.
7038	7	493	0.386975	1	-0.398386	-1.335193	3	11	1995	-0.478022	-1.662698	1.

6887 rows × 15 columns

```
y
```

```
0        850
1        814
2       1729
3       1542
4       1917
        ... 
7034     503
7035    1941
7036    2413
7037    1909
7038    1447
Name: Car_Price, Length: 6887, dtype: int64
```

❖ Now, the next step is **Scaling the data** through **Standard Scaler:**

Scaling the data using the standard Scaler :

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x
```

	Brand	Model	Transmission	Fuel Type	Location	Car_name	Purchase_month	Purchase_year	Kilometers_Driven	owner	Year_of_insurance	M
0	0.525282	0.712804	0.386975	-1.327197	-0.398386	0.830679	0.536465	-0.573051	1.125667	-0.478022	0.525504	
1	0.525282	1.249149	0.386975	-1.327197	-0.398386	0.830679	-0.073060	-1.015446	1.359913	-0.478022	0.525504	
2	0.525282	-0.934541	0.386975	-1.327197	-0.398386	-0.475183	-0.377823	-0.130656	0.927275	-0.478022	-1.662698	
3	0.525282	-0.934541	0.386975	-1.327197	-0.398386	-0.475183	1.145991	-0.573051	1.120886	2.091953	0.525504	
4	0.525282	1.287459	0.386975	0.722221	-0.398386	-0.475183	0.536465	0.754133	-0.141173	-0.478022	0.525504	
...	...	...	...	...	...	...	...	...	...	...	...	
6882	0.525282	-0.140969	0.386975	0.722221	-0.398386	-0.606943	-0.682585	1.196527	-1.236711	2.091953	0.525504	
6883	-1.066367	-1.996284	0.386975	0.722221	-0.398386	-1.335193	-0.377823	1.196527	-0.423224	-0.478022	0.525504	
6884	-1.265323	-1.821151	0.386975	0.722221	-0.398386	1.326082	1.755516	1.638922	-1.287703	-0.478022	0.525504	
6885	-1.066367	-0.097186	0.386975	0.722221	-0.398386	-1.335193	-1.292111	0.311738	-1.676519	-0.478022	0.525504	
6886	-1.066367	0.521252	0.386975	0.722221	-0.398386	-1.335193	-0.682585	0.754133	-0.242361	-0.478022	-1.662698	

6887 rows × 15 columns

❖ Now, here I import few libraries required for training the model and testing and also for **checking the random_state:**

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Checking the random_state:

```
maxAccu=0
maxRS=0
for i in range(1,300):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state=i)
    mod = LinearRegression()
    mod.fit(x_train, y_train)
    pred = mod.predict(x_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Maximum r2 score is ",maxAccu," on Random_state ",maxRS)
```

```
Maximum r2 score is  0.8727643742615423  on Random_state  157
```

Documentation : Here i got 87% for Random_state 157

❖ Here, I will **split the data** into train data and test data through **“train_test_split”** method:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```

❖ Here I have splitted the data with the resulted random_state.

❖ Now, I will **test the data** and find accuracy of the data with different models:

Regression Algorithms :

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

Observation : Here we can see that we have achieved 85.6% accuracy with Linear regression model

Random Forest Regressor:

```
RFR=RandomForestRegressor()
RFR.fit(x_train,y_train)

predRFR=RFR.predict(x_test)
print('R2_Score:',metrics.r2_score(y_test,predRFR))
```

```
R2_Score: 0.9872412713553679
```

```
print(metrics.mean_absolute_error(y_test, predRFR))

print(metrics.mean_squared_error(y_test, predRFR))

print(np.sqrt(metrics.mean_squared_error(y_test, predRFR)))
```

```
17.22074987905177
9409.41900488631
97.00215979495667
```

Linear Regression :

```
LR = LinearRegression()
LR.fit(x_train,y_train)

predLR=LR.predict(x_test)
print('R2_score:',metrics.r2_score(y_test,predLR))
```

```
R2_score: 0.8727643742615423
```

```
print(metrics.mean_absolute_error(y_test, predLR))

print(metrics.mean_squared_error(y_test, predLR))

print(np.sqrt(metrics.mean_squared_error(y_test, predLR)))
```

```
184.39693171244616
93834.84422844413
306.32473655982164
```

Decision Tree Regressor :

```
DTR=DecisionTreeRegressor()
DTR.fit(x_train,y_train)

predDTR=DTR.predict(x_test)
print('R2_Score:',metrics.r2_score(y_test,predDTR))
```

```
R2_Score: 0.9739041406913749
```

```
print(metrics.mean_absolute_error(y_test, predDTR))

print(metrics.mean_squared_error(y_test, predDTR))

print(np.sqrt(metrics.mean_squared_error(y_test, predDTR)))
```

```
23.25399129172714
19245.402999516205
138.72780182615236
```

KNN regressor :

```
knn=KNN()
knn.fit(x_train,y_train)

predknn=knn.predict(x_test)
print('R2_Score:',metrics.r2_score(y_test,predknn))
```

```
R2_Score: 0.78347296305186
```

```
print(metrics.mean_absolute_error(y_test, predknn))

print(metrics.mean_squared_error(y_test, predknn))

print(np.sqrt(metrics.mean_squared_error(y_test, predknn)))
```

```
289.16003870343496
159686.25662312534
399.60762833450184
```

Documentation : Random Forest Regressor has the highest accuracy Score

❖ Now, here I will **check the Cross Validation Scores** of the models tested:

Checking the Cross Validation Score :

```
from sklearn.model_selection import cross_val_score
```

```
# Checking cv score for Random Forest Regression
print(cross_val_score(RFR,x,y,cv=5).mean())
```

```
0.9704807913559954
```

```
# Checking cv score for Linear Regression
print(cross_val_score(LR,x,y,cv=5).mean())
```

```
0.7840686155187255
```

```
# Checking cv score for Decision Tree Regression
print(cross_val_score(DTR,x,y,cv=5).mean())
```

```
0.9618019212661185
```

```
# Checking cv score for KNN Regression
print(cross_val_score(knn,x,y,cv=5).mean())
```

```
0.739460720177331
```

Documentation : Here i can see that among all the models Random Forest Regressor has high CV Score

❖ Here the **"Random Forest model"** has the **highest Cross Validation Score** and so now we will move to **Hyper Parameter Tuning** of the model:

Hyper parameter Tuning :

```
from sklearn.model_selection import GridSearchCV
```

```
parameters = {'criterion':['mse', 'mae'],
              'max_features':['auto', 'sqrt', 'log2'],
              'n_estimators':[0,300],
              'max_depth':[2,4,6,8]}
```

```
GCV=GridSearchCV(RandomForestRegressor(),parameters,cv=5)
```

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

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'criterion': ['mse', 'mae'], 'max_depth': [2, 4, 6, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [0, 300]})
```

```
GCV.best_params_
```

```
{'criterion': 'mse',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 300}
```

```
Cars_model = RandomForestRegressor(criterion='mse', max_depth=8, max_features='auto', n_estimators=300)
Cars_model.fit(x_train, y_train)
pred = Cars_model.predict(x_test)
print("RMSE value:",np.sqrt(metrics.mean_squared_error(y_test, predRFR)))
print('R2_Score:',r2_score(y_test,pred)*100)
```

```
RMSE value: 97.00215979495667
R2_Score: 98.43466262615375
```

Documentation : Here our model has accuracy score of 98.4%

❖ Now, I am going to **Save the model** and also print the **Predicted and Original values of our data:**

Saving the model :

```
import pickle
filename='Cars.pkl'
pickle.dump(RFR,open(filename,'wb'))
```

Conclusion :

```
loaded_model = pickle.load(open('Cars.pkl','rb'))

result=loaded_model.score(x_test,y_test)
result
```

```
0.9872412713553679
```

```
conclusion = pd.DataFrame([loaded_model.predict(x_test)[:],pred[:]],index=["Predicted","Original"])
conclusion
```

	0	1	2	3	4	5	6	7	8	9	10	
Predicted	1186.840000	509.110000	2683.680000	2950.000000	1003.91000	2189.380000	772.03000	1232.570000	356.260000	1071.800000	2318.040000	1661.05
Original	1185.655044	512.823047	2683.590871	2950.138869	1001.25584	2190.698944	775.92107	1507.871093	353.966483	1072.438404	2317.001075	1662.04

✓ Now, here my **best model** is **"Random Forest model"** and is with an accuracy of **98.7%.**

❖ CONCLUSION:

- I have successfully built my model using multiple algorithms and found that the Random Forest Regressor model is the best model.
- As i can see from the boxplot, I couldn’t remove all the outliers but since the data is expensive, I have to proceed with the dataset with outliers.

❖ LIMITATIONS AND SCOPE FOR THE FUTURE:

- ✓ Here the model had more outliers even after treating them and also skewness was also present in few features but we had to proceed and also I was not sure that whether the model can be built well completely after turning into a new dataset.
- ✓ During the data collection I have faced a problem which is few websites, pages do not provide proper information or mixed information which creates problem while building the model.


Thank
You