FLIP ROBO

CAR PRICING

Submitted by:

JASMINE FERNANDES

ACKNOWLEDGMENT

My lecture notes and codes taught in class, You tube

INTRODUCTION

- ## Business Problem Framing

    The client is facing challenges with their past car price valuation machine learning models. The client is looking to find new insights and build new machine learning models from new data.

- ## Conceptual Background of the Domain Problem

- The data is taken from popular car selling websites such as Olx, cardekho, Cars24

- With the covid 19 impact in the car market, there are a lot of changes, some cars are in demand which make them costly and the ones that are not in demand are cheaper than the rest.

- The purpose is to highlight the key correlations impacting price in the newly created data using web scraping. This data is selected based on the key features that would likely impact the price of the car from these three websites

    I have taken features, such as Brand & Model, Varient, Fuel Type, Driven Kilometers , Transmission, Owner, Location, Date of Posting Ad, Price (in ₹)

- ## Review of Literature

- The data is taken from popular car selling websites such as Olx, cardekho, Cars24.The data comprises of 5000+ used car data

- ## Motivation for the Problem Undertaken

- With the covid 19 impact in the car market, there are a lot of changes, some cars are in demand which make them costly and the ones that are not in demand are cheaper than the rest.

- The purpose is to highlight the key correlations impacting price in the newly created data using web scraping. This data is selected based on the key features that would likely impact the price of the car from these three websites

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

- Data Sources and their formats

  Following formats are used:
  ✔ Scatter plots
  ✔ Correlation plot
  ✔ Skewness table
  ✔ Used histogram to view distribution of all the numeric variables
  ✔ seaborn as sns
  ✔ Pie chart with percentage
  ✔ Group by plots
  ✔ dist plot
  ✔ Box plot
  R2 score
  Linear regression
  Random forest
  Deciskn Tree
  Adaboost

- Data Preprocessing Done

  1. Found unique values in object data types, float data types and int data type

20]:
```python
#finding unique values in object data types
def explore_object_type(df,feature_name):
    if df[feature_name].dtype == 'object':
        print(df[feature_name].value_counts())
```

21]:
```python
for featureName in df:
    if df[featureName].dtype == 'object':
        print('\n"' + str(featureName) + '\'s" Values with count are :')
        explore_object_type(df, str(featureName))
```

```
"Brand & Model's" Values with count are :
2013 Maruti Swift              173
2014 Maruti Swift              169
2015 Maruti Swift              142
2018 Maruti Alto 800            84
2017 Hyundai Grand i10          82
                              ...
honda others (2017)              1
mahindra scorpio (2011)          1
maruti suzuki swift (2012)       1
maruti suzuki esteem (2005)      1
hyundai verna (2013)             1
Name: Brand & Model, Length: 382, dtype: int64

"Varient's" Values with count are :
['VDI']        364
['1.2']        357
```

2. In the transmission & fuel type column, there were repeated values.

For example, Petrol was in small case and Capital case. I have treated the data

93]:
```python
# Replacing 'DIESEL' with 'Diesal'
#Replacing PETROL & petrol

df["Fuel Type"] = df["Fuel Type"].replace("DIESEL", "Diesel")
df["Fuel Type"] = df["Fuel Type"].replace("PETROL", "Petrol")
df["Fuel Type"] = df["Fuel Type"].replace("CNG & HYBRIDS", "CNG & Hybrids")
```

39]:
```python
#Manual  & MANUAL is the same
#Automatic & AUTOMATIC are the same

df["Transmission"] = df["Transmission"].replace("MANUAL", "Manual")
df["Transmission"] = df["Transmission"].replace("AUTOMATIC", "Automatic")
```

[56]:
```python
df.info()
```

]:
```python
#replace 1st Owner with First Owner
#replace 2nd Owner with Second Owner
#replace 3rd Owner with Third Owner
#replace Fourth Owner with 4th Owner
#replace #NAME?  with unknown
df["Owner"] = df["Owner"].replace("1st Owner", "First Owner")
df["Owner"] = df["Owner"].replace("2nd Owner", "Second Owner")
df["Owner"] = df["Owner"].replace("3rd Owner", "Third Owner")
df["Owner"] = df["Owner"].replace("4th Owner", "Fourth Owner")
df["Owner"] = df["Owner"].replace("#NAME?", "unknown")
```

Converted Driven Kilometers into float type data as there were numericals in it. Howver, the problem was that KM was attached to them. For example: date read like 24KM. So, even if the data is considered numeric, it was still identified by the system as string or object type data.

I removed: KM, Km, Km, ‘,’ and other special charcaters from the data

```
]: #converting Driven Kilometers to numerical as it is listed in object type of data as listed above
   try:
       df['Driven Kilometers'] = df['Driven Kilometers'].astype(float)
   except ValueError as ve:
       print (ve)
```

```
]: #checking the data for Driven Kilometers
   import seaborn as sns
   location=sns.countplot(x="Driven Kilometers", data=df)
   print(df["Driven Kilometers"].value_counts())
```

```
17000.0     40
90000.0     32
438610.0    29
145000.0    27
120000.0    24
            ..
50.0         1
55520.0      1
8748.0       1
78000.0      1
612310.0     1
Name: Driven Kilometers, Length: 755, dtype: int64
```

```
In [ ]: #removing unique features in Driven Kilometers- removing 'KM'
        df['Driven Kilometers'] = df['Driven Kilometers'].str.replace('KM', '')
```

```
In [ ]: #removing unique features in Driven Kilometers- removing 'Km'
        df['Driven Kilometers'] = df['Driven Kilometers'].str.replace('Km', '')
```

```
In [ ]: #removing unique features in Driven Kilometers- removing '.'
        df['Driven Kilometers'] = df['Driven Kilometers'].str.replace('.', '')
```

```
In [ ]: #removing unique features in Driven Kilometers- removing 'km'
        df['Driven Kilometers'] = df['Driven Kilometers'].str.replace('km', '')
```

```
In [ ]: try:
            df['Driven Kilometers'] = df['Driven Kilometers'].astype(float)
        except ValueError as ve:
            print (ve)
```

- we can see varient feature coulmn is not balanced. we can substitute with mode or remove the null vaues completely. I choose to remove the null values to maintsin accuracy of the data

```
: #as we can see varient is not balanced
  #as it is a small amount, we can substitute with mode or remove completely
  drop_na = ["Varient"]

  for i in drop_na:
      print (i, ":", round((df[i].isna().sum()/df.shape[0])*100, 2))
```

```
Varient : 0.77
```

```
: df = df.dropna(subset=drop_na, axis=0)
```

- Converted date into date time data to use them in graphs, however, dropped date column before data modelling. Used the mode mthod to fill in the null values

```
In [415]: #coverting date into date time data type
          df['Date of Posting Ad']=pd.to_datetime(df['Date of Posting Ad'])
          df.head()
```

```
File C:\anaconda 2022\lib\site-packages\pandas\_libs\tslib.pyx:381, in pandas._libs.tslib.array_to_datetime()

File C:\anaconda 2022\lib\site-packages\pandas\_libs\tslib.pyx:613, in pandas._libs.tslib.array_to_datetime()

File C:\anaconda 2022\lib\site-packages\pandas\_libs\tslib.pyx:751, in pandas._libs.tslib._array_to_datetime_object()

File C:\anaconda 2022\lib\site-packages\pandas\_libs\tslib.pyx:742, in pandas._libs.tslib._array_to_datetime_object()

File C:\anaconda 2022\lib\site-packages\pandas\_libs\tslibs\parsing.pyx:281, in pandas._libs.tslibs.parsing.parse_datetime_s
tring()

File C:\anaconda 2022\lib\site-packages\dateutil\parser\_parser.py:1368, in parse(timestr, parserinfo, **kwargs)
   1366     return parser(parserinfo).parse(timestr, **kwargs)
   1367 else:
-> 1368     return DEFAULTPARSER.parse(timestr, **kwargs)

File C:\anaconda 2022\lib\site-packages\dateutil\parser\_parser.py:643, in parser.parse(self, timestr, default, ignoretz, tz
infos, **kwargs)
    640 res, skipped_tokens = self._parse(timestr, **kwargs)
```

```
In [417]: #removing unique values in the Date of Posting Ad that are not dates
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Maharashtra', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Delhi', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Rajasthan', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('WB', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Other minor issues', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Haryana', '')
          df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('AP', '')
```

```
410]: #as we can see varient is not balanced
      #as it is a small amount, we can substitute with mode or remove completely
      drop_na = ["Varient"]

      for i in drop_na:
          print (i, ":", round((df[i].isna().sum()/df.shape[0])*100, 2))

      Varient : 0.77
```

```
411]: df = df.dropna(subset=drop_na, axis=0)
```

```
412]: df.info()

      <class 'pandas.core.frame.DataFrame'>
      Int64Index: 5011 entries, 0 to 5049
      Data columns (total 9 columns):
       #   Column              Non-Null Count  Dtype
      ---  ------              --------------  -----
       0   Brand & Model       5011 non-null   object
       1   Varient             5011 non-null   object
       2   Fuel Type           5011 non-null   object
       3   Driven Kilometers   5011 non-null   float64
       4   Transmission        5011 non-null   object
       5   Owner               5011 non-null   object
       6   Location            5011 non-null   object
       7   Date of Posting Ad  5011 non-null   object
       8   Price (in ₹)        5011 non-null   int64
      dtypes: float64(1), int64(1), object(7)
      memory usage: 391.5+ KB
```

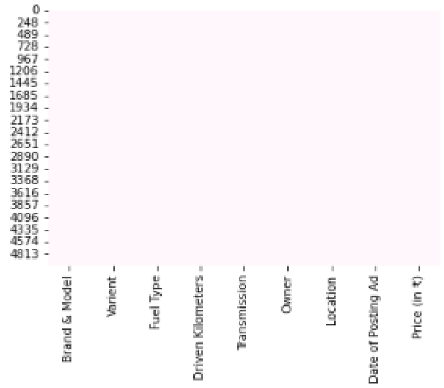
Checked for any missing values:

367]:

```
#checking for any missing data
# Missing Data Pattern
import seaborn as sns
sns.heatmap(df.isnull(), cbar=False, cmap='PuBu')
```

367]:

```
<AxesSubplot:>
```



368]:

```
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing.head()
#as we can see, there are no missing values
```

368]:

	Total	Percent
Brand & Model	0	0.0
Varient	0	0.0
Fuel Type	0	0.0
Driven Kilometers	0	0.0
Transmission	0	0.0

- # Data Inputs- Logic- Output Relationships

- **ad**

7]:

```
#removing unique values in the Date of Posting Ad that are not dates
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Maharashtra', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Delhi', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Rajasthan', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('WB', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Other minor issues', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('Haryana', '')
df['Date of Posting Ad'] = df['Date of Posting Ad'].str.replace('AP', '')
```

8]:

```
df['Date of Posting Ad']=pd.to_datetime(df['Date of Posting Ad'])
df.head()
```

8]:

	Brand & Model	Varient	Fuel Type	Driven Kilometers	Transmission	Owner	Location	Date of Posting Ad	Price (in ₹)
0	Mahindra Xuv500 (2013)	W8 Dual Tone	Diesel	58000.0	Manual	First Owner	Pitampura, Delhi	2022-01-27	435000
1	Hyundai Creta (2020)	1.6 SX Option Executive Diesel	Diesel	438810.0	Manual	First Owner	Ahiritola, Kolkata	2022-01-23	1165101
2	Hyundai Verna (2019)	VTVT 1.4 EX	Petrol	17000.0	Manual	Second Owner	Chelavoor, Pantheeramkavu	2022-01-25	815000
3	Datsun Redigo (2020)	D	Petrol	10000.0	Manual	First Owner	Palam, Delhi	2022-01-13	270000
4	Hyundai I10 (2011)	Sportz 1.1 iRDE2	Petrol	70000.0	Manual	First Owner	Dwarka Sector 13, Delhi	2022-01-13	185000

0]:

```
#using mode to replace null values
df['Date of Posting Ad'] = df['Date of Posting Ad'].fillna(df['Date of Posting Ad'].mode()[0])
```

- **With the transpose method , we can conclude the following insights**

#we can observe that 2013 Maruti Swift is the best brand and model

#The most popular varient is ['VDI']

#The most popular location is Chelavoor, Pantheeramkavu to buy cars

#The most popular Date of Posting Ad is 2021-02-27(27th Feb 2021

```
df.describe(include=['object','datetime']).transpose()
```

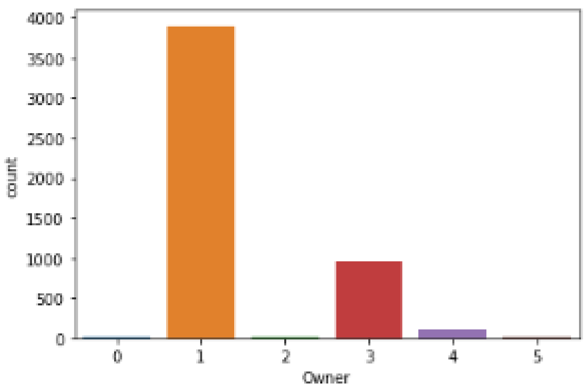
	count	unique	top	freq	first	last
Brand & Model	5011	374	2013 Maruti Swift	173	NaT	NaT
Varient	5011	345	['VDI']	364	NaT	NaT
Fuel Type	5011	6	Petrol	2611	NaT	NaT
Transmission	5011	13	Manual	4299	NaT	NaT
Owner	5011	6	First Owner	3898	NaT	NaT
Location	5011	185	Chelavoor, Pantheeramkavu	210	NaT	NaT
Date of Posting Ad	5011	44	2022-01-27 00:00:00	996	2021-02-27	2022-12-01

```
#we can observe that 2013 Maruti Swift is the best brand and model
#The most popular varient is ['VDI']
#The most popular location is Chelavoor, Pantheeramkavu to buy cars
#The most popular Date of Posting Ad is 2021-02-27(27th Feb 2021)
```

- As we can see the maximum owners own 1st hand cars

```
#checking for newly named values
import seaborn as sns
location=sns.countplot(x="Owner", data=df)
print(df["Owner"].value_counts())
#as we can see the maximum owners own 1st hand cars
```

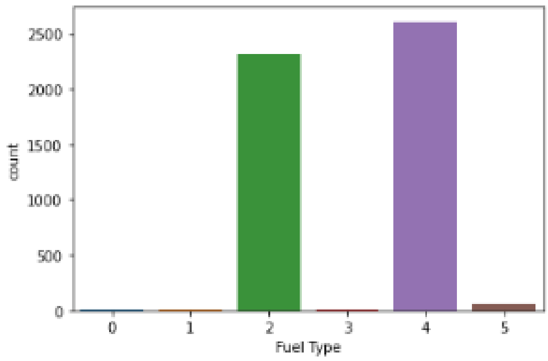
```
1    3898
3     953
4     104
2      24
0      20
5      12
Name: Owner, dtype: int64
```



- As we can see the maximum car owners use petrol

```
7]: #checking f0r newly named values
import seaborn as sns
location=sns.countplot(x="Fuel Type", data=df)
print(df["Fuel Type"].value_counts())
#as we can see the maximum car owners use petrol
```

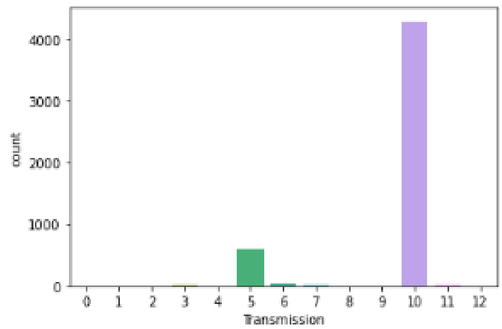
```
4    2611
2    2315
5      55
0      13
1      10
3       7
Name: Fuel Type, dtype: int64
```



- As we can see the maximum car owners prefer manual transmissions

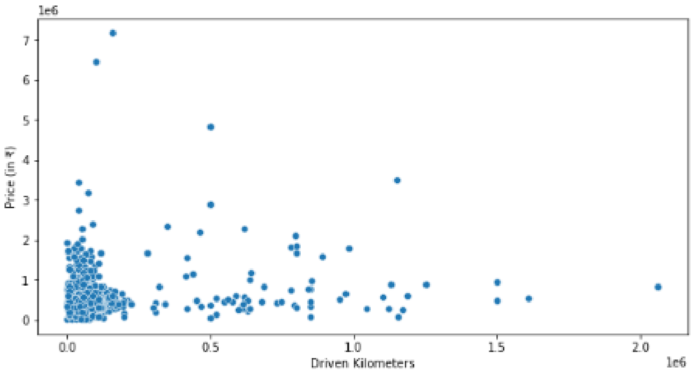
```
In [448]: #checking f0r newly named values
import seaborn as sns
location=sns.countplot(x="Transmission", data=df)
print(df["Transmission"].value_counts())
#as we can see the maximum car owners prefer manual transmissions
```

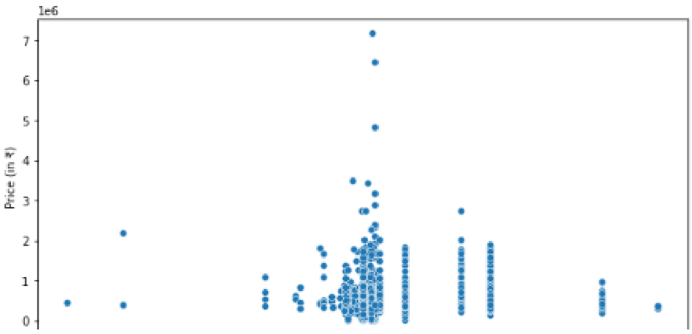
```
10    4299
5      600
6       31
3       23
7       12
11       9
12       6
4        6
8        6
2        6
0        6
9        5
1        2
Name: Transmission, dtype: int64
```



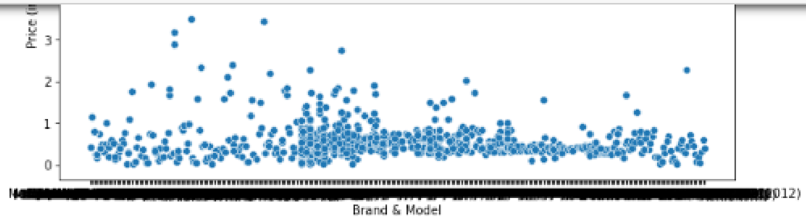
Comparing all other features with Sales

26]:
```python
#measuring sales relation with all other columns:
for col in df.columns:
    if df[col].dtype != 'object':
        plt.figure(figsize=(10,5))
        sns.scatterplot(x=col,y='Price (in ₹)',data=df)
```

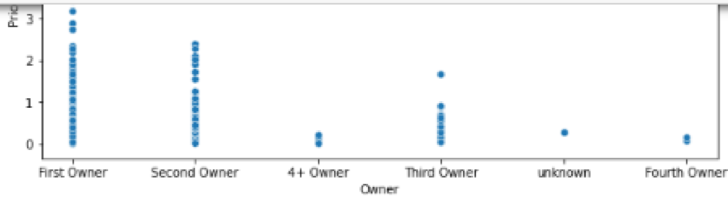




```
#measuring sales relation with all other columns:
for col in df.columns:
    if df[col].dtype != 'float64':
        plt.figure(figsize=(10,5))
        sns.scatterplot(x=col,y='Price (in ₹)',data=df)
```

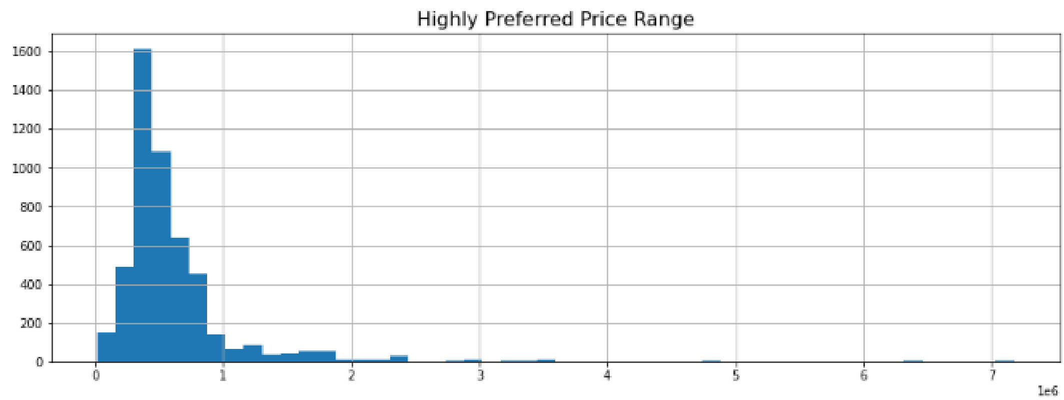



```
#measuring sales relation with all other columns:
for col in df.columns:
    if df[col].dtype != 'int64':
        plt.figure(figsize=(10,5))
        sns.scatterplot(x=col,y='Price (in ₹)',data=df)
```



1e6

```
df["Price (in ₹)"].hist(bins=50, figsize=(15, 5))
plt.title("Highly Preferred Price Range", fontsize=16);
```



Use the label encoder to convert object data types into numerical

```
#label ensoding
from sklearn.preprocessing import LabelEncoder
for col in df.columns:
    if df[col].dtypes == 'object':
        encoder = LabelEncoder()
        df[col] = encoder.fit_transform(df[col])
```

```
df.head()
```

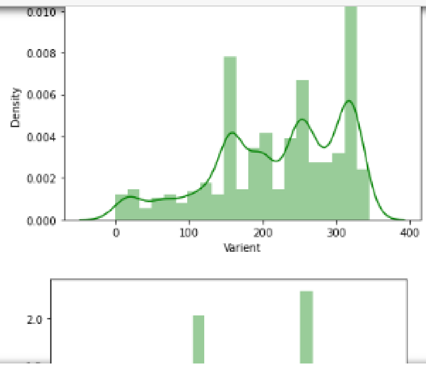
Driven Kilometers & Brand & Model have the highest correlation with Price

Brand & Model	Varient	Fuel Type	Driven Kilometers	Transmission	Owner	Location	Price (in ₹)

```
[ ]: #RELATION WITH Price
     corr_matrix=df.corr()
     corr_matrix["Price (in ₹)"].sort_values(ascending=False)

[ ]: Price (in ₹)           1.000000
     Driven Kilometers      0.195761
     Brand & Model          0.175108
     Location              -0.027278
     Owner                 -0.033436
     Fuel Type             -0.048455
     Varient               -0.139465
     Transmission          -0.250432
     Name: Price (in ₹), dtype: float64

[ ]: #Driven Kilometers &  Brand & Model have the hghest correlation with Price
```

Checking for outliers and skewness in data: I found Driven Kilometers & Transmission have maximum skewness

```
In [437]: #checking skewness

          for col in df.describe().columns:
              sns.distplot(df[col],color='g')
              plt.show()
```




```
In [438]: df.skew()

Out[438]: Brand & Model         0.652163
          Varient              -0.606563
          Fuel Type            -0.146462
          Driven Kilometers     4.763624
          Transmission         -2.243327
          Owner                 1.526215
          Location              0.029933
          Price (in ₹)          5.309548
          dtype: float64
```

- ## Hardware and Software Requirements and Tools Used

  Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

I removed outliers and skewness with Power Tranform & Standard Scalar method

```
In [485]: #Transforming data to remove skewness:
          from sklearn.preprocessing import power_transform
          X_new=power_transform(X)
```

```
In [486]: from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import StandardScaler

          from sklearn.preprocessing import StandardScaler
          sc = StandardScaler()
          Scaled_X=sc.fit_transform(X_new)
```

```
In [487]: #Transforming data to remove skewness:
          Scaled_X

Out[487]: array([[ 1.38268478, -0.9374429 , -1.04163908, ...,  0.35945409,
                  -0.44168113,  0.62907684],
                 [ 1.20053657, -2.06147148, -1.04163908, ...,  0.35945409,
                  -0.44168113, -1.95870854],
                 [ 1.29662774, -1.07665349,  0.9068391 , ...,  0.35945409,
                   1.6864798 , -1.0303697 ],
                 ...,
                 [ 1.97347924, -1.16066209,  0.9068391 , ..., -2.35768298,
                  -0.44168113,  0.67593671],
                 [ 1.98693563, -1.0660615 ,  0.9068391 , ...,  0.35945409,
                  -0.44168113,  0.03585368],
                 [ 2.09318929, -1.04481816, -1.04163908, ...,  0.35945409,
                  -0.44168113, -0.12338528]])
```

- Working with linear regression to get the accuracy score

```
In [492]: #using linear regression
          from sklearn import metrics
          from sklearn.linear_model import LinearRegression
          from sklearn.model_selection import train_test_split
          lr=LinearRegression()
```

```
In [494]: from sklearn.metrics import r2_score
          for i in range(0,300):
              x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,random_state=i)
              lr.fit(x_train,y_train)
              pred_train=lr.predict(x_train)
              pred_test=lr.predict(x_test)
              print(f"At random state {i},the training accuracy is:- {r2_score(y_train, pred_train)}")
              print(f"At random state {i},the testing accuracy is:- {r2_score(y_test,pred_test)}")
              print("\n")
```

```
At random state 0,the training accuracy is:- 0.0830003596528609
At random state 0,the testing accuracy is:- 0.09372873456478148


At random state 1,the training accuracy is:- 0.0857322932883764
At random state 1,the testing accuracy is:- 0.083602898872568


At random state 2,the training accuracy is:- 0.07263114797612436
At random state 2,the testing accuracy is:- 0.10896616827219041


At random state 3,the training accuracy is:- 0.09885236434100131
At random state 3,the testing accuracy is:- 0.05262613172063479 4


At random state 4,the training accuracy is:- 0.0922709391748584
At random state 4,the testing accuracy is:- 0.057988692119015584
```

Splitting the test and train data

```
In [496]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=15)
```

```
In [497]: lr=LinearRegression()
          lr.fit(x_train,y_train)
          pred=lr.predict(x_test)
```

```
In [498]: print('MAE:', metrics.mean_absolute_error(y_test, pred))
          print('MSE:', metrics.mean_squared_error(y_test, pred))
          print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))

MAE: 284479.7904573079
MSE: 260937763871.29694
RMSE: 510820.6768243597
```

```
In [499]: lr.score(x_train , y_train)
Out[499]: 0.07515133483565051
```

Accuracy of decision Tree - 99%

```
In [500]: #with descision tree
          from sklearn.tree import DecisionTreeRegressor
          dt=DecisionTreeRegressor()
          dt.fit(x_train,y_train)
          predD=dt.predict(x_test)
```

```
In [501]: print('MAE:', metrics.mean_absolute_error(y_test, predD))
          print('MSE:', metrics.mean_squared_error(y_test, predD))
          print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predD)))

MAE: 64756.69672785315
MSE: 43932153838.792496
RMSE: 209599.9853024625
```

```
In [502]: dt.score(x_train , y_train)
Out[502]: 0.9971448915816747
```

Score for Random Forest - 98%

In [503]:
```
#With random forest
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)
predR=rfr.predict(x_test)
```

In [504]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predR))
print('MSE:', metrics.mean_squared_error(y_test, predR))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predR)))
```

```
MAE: 84164.98196397908
MSE: 29417985281.05871
RMSE: 171516.72012098035
```

In [505]:
```
rfr.score(x_train , y_train)
```

Out[505]: 0.9852459906128679

Sore for Adaboost - 42%

In [506]:
```
#with Adaboost
from sklearn.ensemble import AdaBoostRegressor
ada = AdaBoostRegressor()
ada.fit(x_train,y_train)
predA=ada.predict(x_test)
```

In [507]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predA))
print('MSE:', metrics.mean_squared_error(y_test, predA))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predA)))
```

```
MAE: 293248.0301384547
MSE: 136510861297.20705
RMSE: 369473.76266415324
```

In [508]:
```
ada.score(x_train , y_train)
```

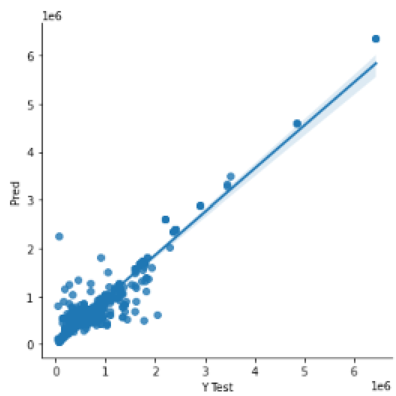
Out[508]: 0.4208804163827402

The regression line- Random Forest works well

In [510]:
```
#with random forest
data = pd.DataFrame({'Y Test':y_test , 'Pred':predR},columns=['Y Test','Pred'])
sns.lmplot(x='Y Test',y='Pred',data=data,palette='rainbow')
data.head()
```

Out[510]:

	Y Test	Pred
4764	303399	382179.03
2937	688699	665282.00
3850	474499	490644.00
2387	395699	413186.00
2438	435399	474632.00



```
In [511]: Random_Forest=rfr.fit(x_train,y_train)

In [512]: import pickle as pkl
          Car_Price_Model = 'Car_Price_Model.pickle'
          pkl.dump(Random_Forest, open(Car_Price_Model,'wb'))

In [ ]: df.
```

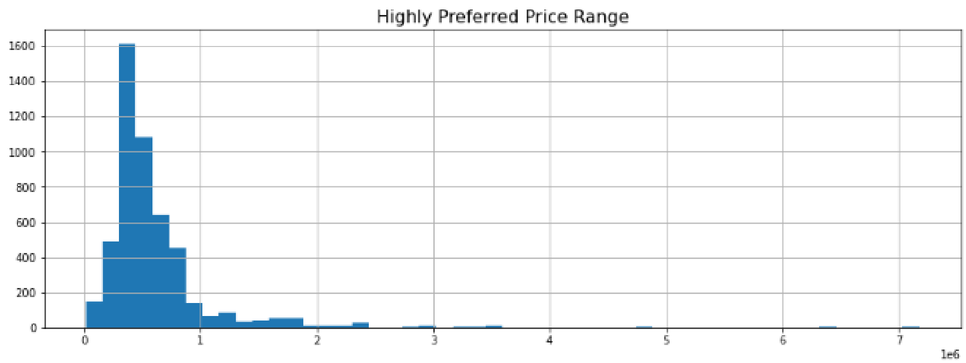
CONCLUSION

- The data had lots of punctuation and similar words in the feature columns which had to be removed as data was scraped from 3 different websites
- The data was not balanced, so null values were removed by different methods.such as mode, replace functions, etc
- After performing power transform and standard scalar, the outliers and skewness was removed to quite an extent

- maximum car owners use petrol

Transmission 0 0.0

In [369]:
```
df["Price (in ₹)"].hist(bins=50, figsize=(15, 5))
plt.title("Highly Preferred Price Range", fontsize=16);
```

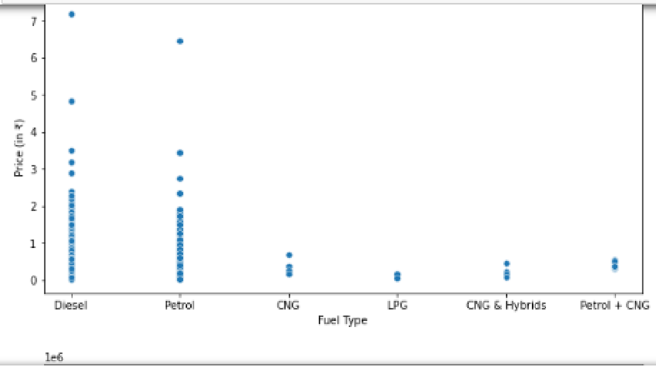


-

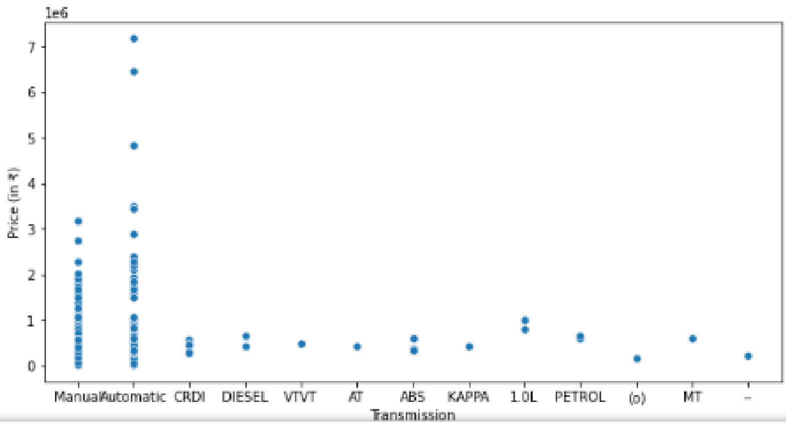
Sales VS other features

Diesel fetches the highest in price in the 'Fuel Type' category

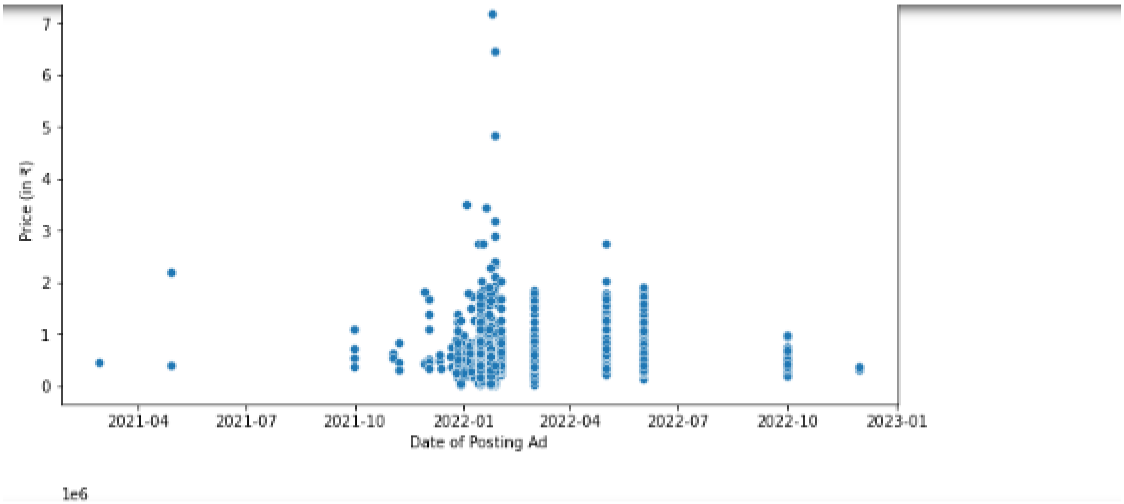
In [430]:
```
#measuring sales relation with all other columns:
for col in df.columns:
    if df[col].dtype != 'float64':
        plt.figure(figsize=(10,5))
        sns.scatterplot(x=col,y='Price (in ₹)',data=df)
```



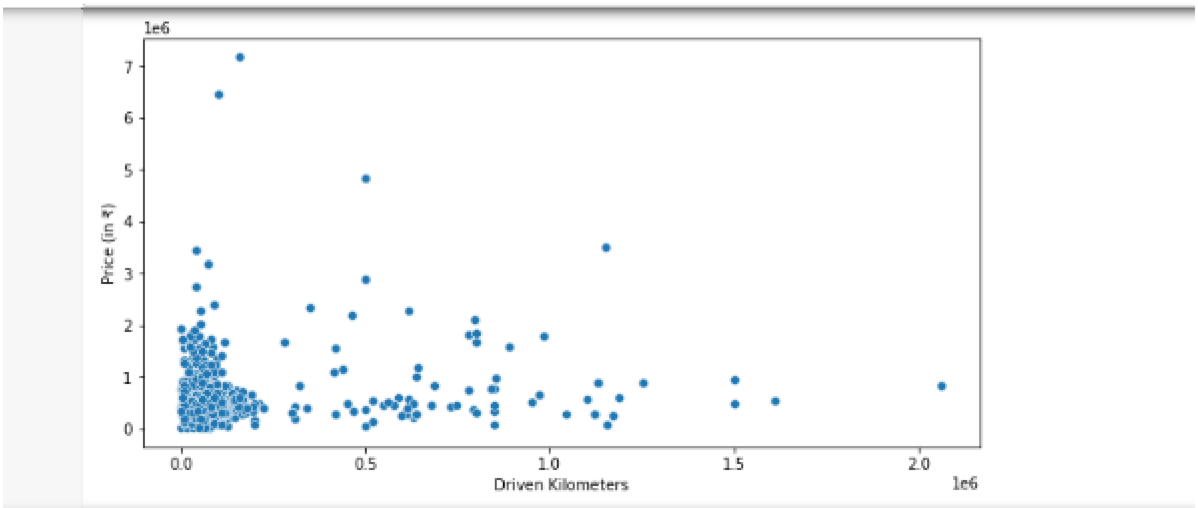
Automatic fetches the highest price in the transmission feature column category



Ads posted in Feb 2022 had the most sales



The cars that were driven the least per kilometer gathered more price



4+ and 4th owner fetches the least price in the market