Car Price Prediction Project

Submitted by:

SHREYA JAIN

ACKNOWLEDGMENT

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. I am also grateful to Mr. Keshav Banal for her constant guidance and support.

INTRODUCTION

- Business Problem Framing

  In this project, we have to make car price valuation model using new machine learning models from new data. Because with the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models.

- Conceptual Background of the Domain Problem
  - Firstly, we will prepare our own dataset using web scraping.
  - After that we will check whether the project is a regression type or a classification type.
  - We will also check whether our dataset is balanced or imbalanced. If it is an imbalanced one, we will apply sampling techniques to balance the dataset.
  - Then we will do model building and check its accuracy.
  - Our main motto is to build a model with good accuracy and for that we will also go for hyperparameter tuning.
- Review of Literature

I am summarizing my research done on the topic.

- I have created my own dataset using web scraping and imported important libraries for my project.
- I have created the data frame.
- I have analyzed my data by checking its shape, number of columns, presence of null values if any and checking the datatypes.
- Then I have done some data cleaning steps, e.g., Checking the value counts of the target variable, dropping some irrelevant columns from the dataset, checking correlation between the dependent and independent variables using heatmap, visualizing data using distribution plots, detecting and removing skewness in my data if any, outliers detection using boxplots and removing them, balancing dataset, splitting the data into independent and dependent variables and finally scaling the data.
- Then I have used 9 models, out of which RandomForestRegressor is giving a good accuracy score of 99% after hyperparameter tuning.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  - If you look at data science, we are actually using mathematical models to model (and hopefully through the model to explain some of the things that we have seen) business circumstances, environment etc and through these model, we can get more insights such as the outcomes of our decision undertaken, what should we do next or how shall we do it to improve the odds. So mathematical models are important, selecting the right one to answer the business question can bring tremendous value to the organization.
  - Here I am using RandomForestRegressor with accuracy 98% after hyper parameter tuning.
  - Data Sources and their formats
  - Data Source: The read_csv function of the pandas library is used to read the content of a CSV file into the python environment as a pandas DataFrame. The function can read the files from the OS by using proper path to the file.
  - Data description: Pandas describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values

- Data Preprocessing Done

  - I have checked for null values and there are some null values present, I have removed it using SimpleImputer method.
  - I have label encoded the object type columns in the dataset.
  - I have checked the correlation between dependent and independent variables using heatmap. I have seen most of the independent variables are correlated with each other and the target variable is positively correlated with a very few independent variables.
  - I have done some visualization using histogram**.**
  - I have checked outliers using boxplots, but no outliers are present.
  - I also have checked for skewness in my data, but the skewness present is very negligible, so I don’t consider it.
  - I have splitted the dependent and independent variables into

x and y.

- I have scaled the data using StandardScaler method and made my data ready for model building.

- Hardware and Software Requirements and Tools Used
  - Hardware requirements:

    Processor: AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz

    RAM: 8.00 GB (7.37 GB usable)

    System type: Windows 11 Pro

  - Software requirements:

    Python: One of the most used programming languages Tools used:

    Jupyter notebook: Jupyter is a free, open-source, interactive web tool known as a computational notebook where I have written my python codes.

    NumPy: NumPy is an open-source numerical Python library. NumPy contains a multi-dimensional array and matrix data structures.

    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. Matplotlib: It is a Python library used for plotting graphs with the help of other libraries like Numpy and Pandas. It is a powerful tool for visualizing data in Python.

    Seaborn: It is also a Python library used for plotting graphs with the help of Matplotlib, Pandas, and Numpy.

Scikit-learn: It is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Scipy.stats: This module contains a large number of probability distributions as well as a growing library of statistical functions.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)
  - To check the correlation among the data, I have used **heatmap** to visualize it.
  - To get a clear view of the columns visually, I have used **distribution plots**.
  - For checking outliers, I have used **boxplots.**
  - For scaling the data, I have used **StandardScaler** method.
  - For training and testing the data, I have imported **train_test_split library** from scikit-learn.
  - For model building, I have used 9 models(LassoRegression(), ElasticNetRegression(), RidgeRegression(), DecisionTreeRegressor(), KNeighborsRegressor(), AdaBoostRegressor(),LinearRegression(), RandomForestRegression(), GradientBoostingRegressor()), out of which **RandomForestRegression** model is the best model for my dataset.
  - For better accuracy of the model, I have used **hyperparameter tuning (RandomizedSearchCV).**

- Run and Evaluate selected models

In [68]:
```
#Finding the best random state and r2_score
for i in range(42,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=i)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pred_test)*100,1):
        print('At random state',i,',the model performs well')
        print('Training r2_score is: ',r2_score(y_train,pred_train)*100)
        print('Testing r2_score is: ',r2_score(y_test,pred_test)*100)
```

```
At random state 54 ,the model performs well
Training r2_score is:  70.07096576957927
Testing r2_score is:  70.13238349195949
At random state 58 ,the model performs well
Training r2_score is:  70.08423009151878
Testing r2_score is:  70.10140926126549
At random state 60 ,the model performs well
Training r2_score is:  70.08257376435378
Testing r2_score is:  70.09190767021569
At random state 68 ,the model performs well
Training r2_score is:  70.09454430454532
Testing r2_score is:  70.0619973058046
At random state 83 ,the model performs well
Training r2_score is:  70.09747101108337
Testing r2_score is:  70.05752733796045
At random state 88 ,the model performs well
Training r2_score is:  70.08974187727294
Testing r2_score is:  70.05907571962359
```

In [70]:
```
LR=LinearRegression()
l=Lasso()
en=ElasticNet()
rd=Ridge()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
rf=RandomForestRegressor()
ab=AdaBoostRegressor()
gb=GradientBoostingRegressor()
```

In [71]:
```
models= []
models.append(('Linear Regression',LR))
models.append(('Lasso Regression',l))
models.append(('Elastic Net Regression',en))
models.append(('Ridge Regression',rd))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
models.append(('RandomForestRegressor',rf))
models.append(('AdaBoostRegressor',ab))
models.append(('GradientBoostingRegressor',gb))
```

Hyperparameter Tuning :

Random Forest Regressor

In [74]:
```
#Creating parameter list to pass in GridSearchCV
parameters={'criterion':['mse','mae'],'n_estimators':[50,100,500],'max_features':['auto','sqrt','log2']}
```

In [75]:
```
#Using GridSearchCV to run the parameters and checking final accuracy
rf=RandomForestRegressor()
grid=GridSearchCV(rf,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_) #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'criterion': 'mse', 'max_features': 'auto', 'n_estimators': 50}
0.984574811251329
```

In [76]:
```
#Using the best parameters obtained
RF=RandomForestRegressor(random_state=48, n_estimators=500, criterion='mse', max_features='log2')
RF.fit(x_train,y_train)
pred=RF.predict(x_test)
print('r2_score: ',r2_score(y_test,pred)*100)
print('Standard deviation: ',cross_val_score(RF,x,y,cv=5,scoring='r2').std())
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

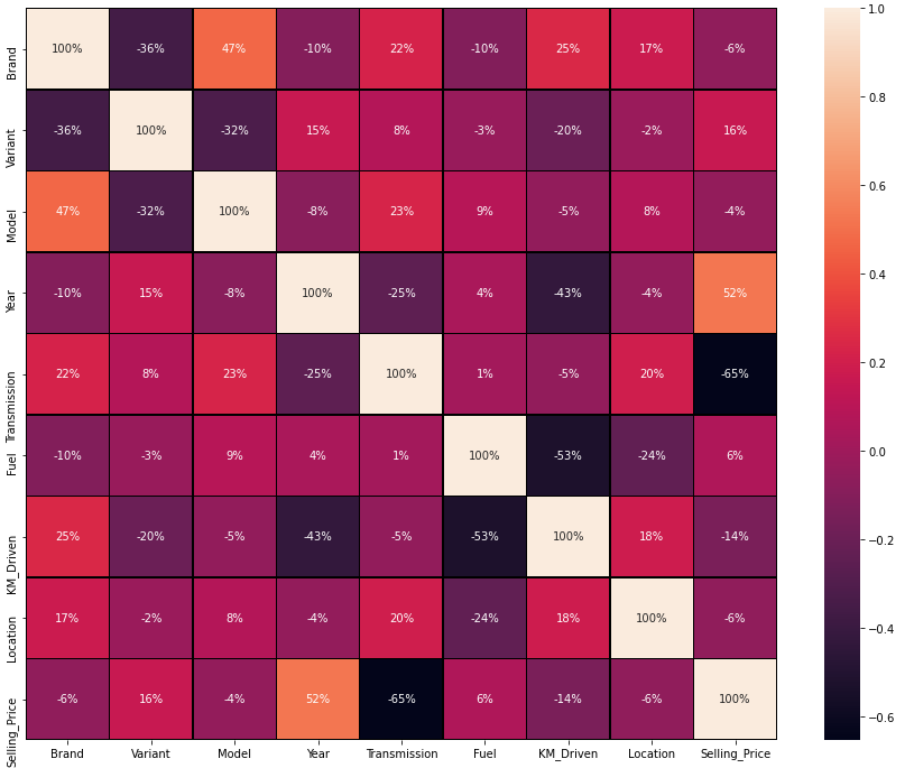
```
r2_score:  98.50456691899329
Standard deviation:  0.00434432139752803
Mean absolute error:  26420.99733115128
Mean squared error:  3753715274.588142
Root Mean squared error:  61267.57114973746
```

After hyperparameter tuning, the accuracy is 98%.

- Visualizations

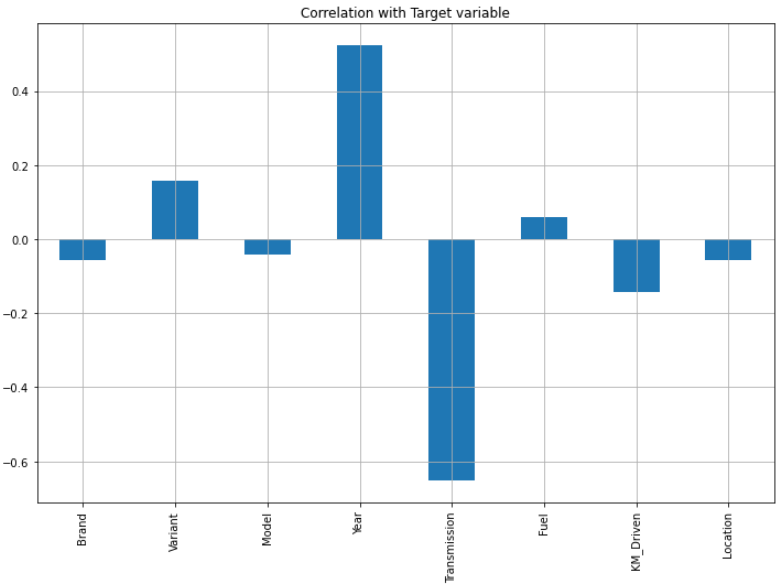
  Correlation matrix using heatmap-checking correlation between dependant and independent variables.

In [51]:
```
#Plotting heatmap for visualizing the correlation
plt.figure(figsize=(15,12))
sn.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',annot=True)
plt.show()
```

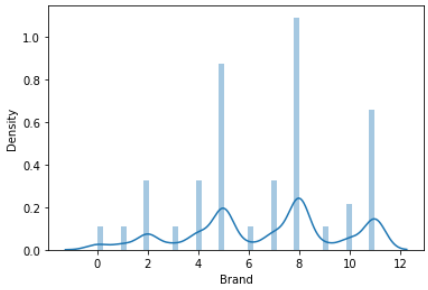


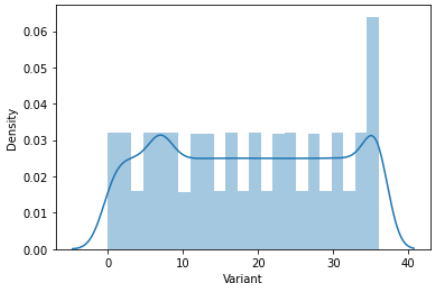
In [52]:
```
#Correlation with target variable
plt.figure(figsize=(12,8))
df.drop('Selling_Price',axis=1).corrwith(df['Selling_Price']).plot(kind='bar',grid=True)
plt.title('Correlation with Target variable')
```

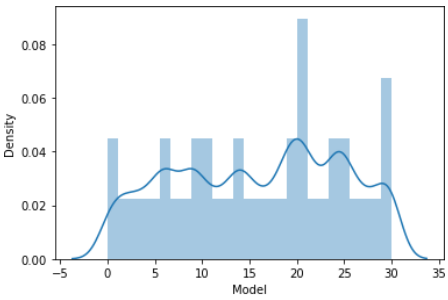
Out[52]: Text(0.5, 1.0, 'Correlation with Target variable')

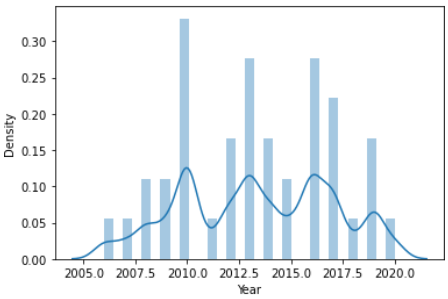


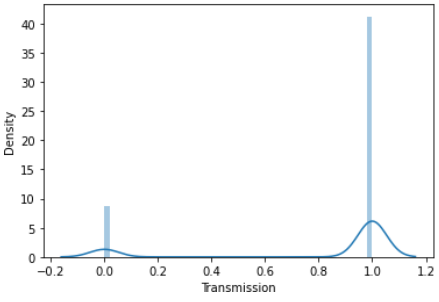
In [54]:
```
#Plotting distplot for checking the distribution of skewness
for col in df.describe().columns:
    sn.distplot(df[col])
    plt.show()
```

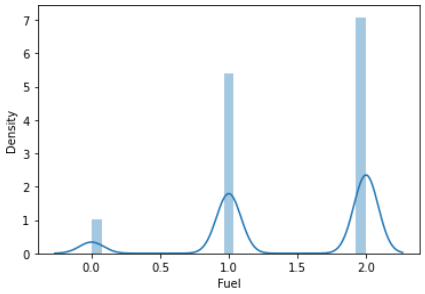


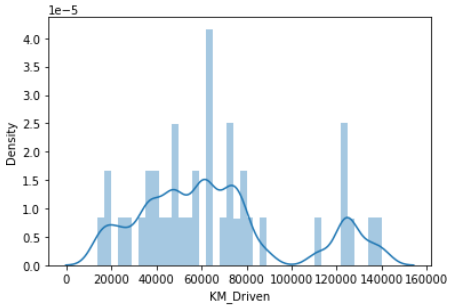


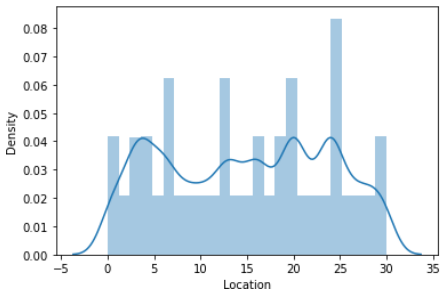


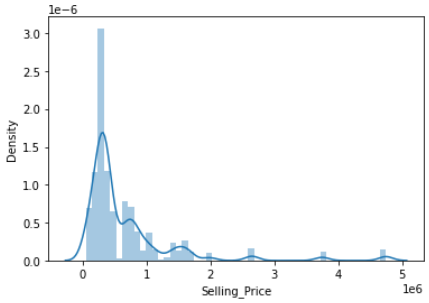






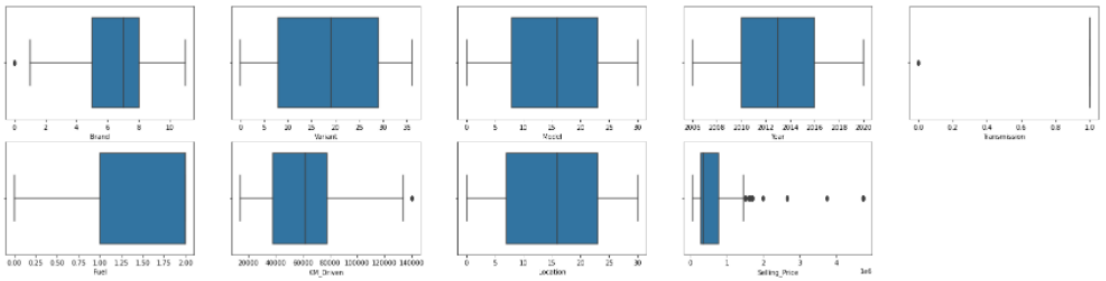






Checking outliers :

In [55]:
```
collist=df.columns.values
ncol=5
nrow=8
plt.figure(figsize=(30,30))
for i in range(0,len(collist)):
    plt.subplot(nrow,ncol,i+1)
    sn.boxplot(df[collist[i]])
```



- Interpretation of the Results
  - In the visualization part, I have seen how my data looks like using heatmap, boxplot, distribution plots, histogram etc.
  - In the pre-processing part, I have cleaned my data using many methods like SimpleImputer,LabelEncoder etc.
  - In the modelling part, I have designed our model using algorithm like **RandomForestRegression**.
  - The accuracy, Mean Absolute Error, Mean Squared Error, Root Mean Absolute Error are achieved for the model.

Project Title : Car Price Prediction Project

Problem Statement:

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model.

Business Goal:

We need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model.

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Loading the data-set
df=pd.read_csv("used_car_data.csv")
df.head(5)
```

Out[2]:

	Unnamed: 0	Brand	Model	Variant	Year	Selling_Price	KM_Driven	Fuel	Transmission	Lo
0	0	Hyundai	i20	Magna Optional 2012-2014 1.2	2013	3,40,000	63,000 km	Petrol	Manual	Delh
1	1	Audi	A6	-	2016	26,50,000	72,000 km	Petrol	Automatic	Nan Tar
2	2	Maruti Suzuki	Alto 800	2012-2016 VXI	2016	2,95,000	37,739 km	Petrol	Manual	M Maha
3	3	Maruti Suzuki	Swift	-	2007	2,85,000	38,000 km	Petrol	Manual	Ka Ben
4	4	Maruti Suzuki	Swift Dzire	1.2 Vxi BSIV	2015	4,70,000	51,000 km	Petrol	Manual	Pree Delh

In [3]:
```
#checking the shape of the data-set
df.shape
```

Out[3]:
```
(18000, 10)
```

Data Pre-Processing :

In [4]:
```
#checking the variant column
df['Variant'].value_counts()
```

Out[4]:
```
-                                  1431
2.8Z Automatic                      896
Petrol HSE Dynamic                  450
2.5 G (Diesel) 8 Seater             450
Sportz (O) 1.4                      450
1.6 SX Plus Auto                    450
Smart Hybrid Alpha                  449
i-VTEC S                            449
1.2 Vxi BSIV                        449
i-VTEC VX                           449
LTZ                                 449
2004-2011 2.5 G4 Diesel 7-seater    449
2012-2015 85PS Diesel RxL           449
5 STR With AC Plus HTR CNG          449
2012 LTZ AT                         449
Asta 1.2 (O)                        449
VXI CNG                             448
GLS                                 448
2011-2014 VDI                       448
1.1 GVS Option                      448
SLX                                 448
1.2 VX                              448
2012-2014 Magna Optional 1.2        448
2006-2010 LXI Minor                 448
320d Sport                          448
2012-2016 VXI                       447
LXI BS IV                           447
V                                   447
Diesel LS                           447
1.2 Magna Executive                 447
1.5 TDCi Titanium Plus              446
2011-2016 4x2 Manual                446
2009-2011 Lx BSIV                   446
2009-2011 E4 BS IV                  446
Era +                               445
2009-2011 Cx BSIV                   444
LXI Option                          443
Name: Variant, dtype: int64
```

Observations :

'Unnamed: 0' is unwanted index column

In [5]:
```
#dropping the unwanted columns
df.drop(['Unnamed: 0'], inplace=True, axis = 1)
df.head(5)
```

Out[5]:

	Brand	Model	Variant	Year	Selling_Price	KM_Driven	Fuel	Transmission	Location

	Brand	Model	Variant	Year	Selling_Price	KM_Driven	Fuel	Transmission	Location
0	Hyundai	i20	2012-2014 Magna Optional 1.2	2013	3,40,000	63,000 km	Petrol	Manual	Geeta Colony, Delhi, Delhi
1	Audi	A6	-	2016	26,50,000	72,000 km	Petrol	Automatic	Nandanam Nandanam, Chennai, Tamil Nadu
2	Maruti Suzuki	Alto 800	2012-2016 VXI	2016	2,95,000	37,739 km	Petrol	Manual	Andheri East, Mumbai, Maharashtra
3	Maruti Suzuki	Swift	-	2007	2,85,000	38,000 km	Petrol	Manual	Banaswadi Kasturi Nagar, Bengaluru, Karnataka
4	Maruti Suzuki	Swift Dzire	1.2 Vxi BSIV	2015	4,70,000	51,000 km	Petrol	Manual	Preet Vihar, Delhi, Delhi

In [6]:
```
##changing the dtype and striping the unwanted commas,words from the data

#Kilometers Driven
df['KM_Driven'] = df['KM_Driven'].str.replace('km','')
df['KM_Driven'] = df['KM_Driven'].str.replace(',','')
df['KM_Driven'] = df['KM_Driven'].str.replace('-','0')

#converting into int
df['KM_Driven'] = df['KM_Driven'].astype(int)
```

In [7]:
```
#finding mean for kms driven
kmsmean = df['KM_Driven'].mean()
kmsmean
```

Out[7]: 65147.55127777778

In [8]:
```
#Replacing 0 with mean value in kms driven
df['KM_Driven'] = df['KM_Driven'].apply(lambda x: x if x!=0 else kmsmean)
```

In [9]:
```
#Price
df['Selling_Price'] = df['Selling_Price'].str.strip()
df['Selling_Price'] = df['Selling_Price'].str.replace(',','')
df['Selling_Price'] = df['Selling_Price'].str.replace('-','0')

#converting into float
df['Selling_Price'] = df['Selling_Price'].astype(float)
```

In [10]:
```
#finding mean for kms driven
pricemean = df['Selling_Price'].mean()
pricemean
```

Out[10]: 662853.2136111112

In [11]:
```
#Replacing 0 with mean value in kms driven
df['Selling_Price'] = df['Selling_Price'].apply(lambda x: x if x!=0 else pricemean)
```

In [12]:
```
#Year_of_Manufacturing
df['Year'] = df['Year'].apply(lambda x: int(x.strip()[0:4]) if x!='-' else 0)

# median year
median_year=df['Year'].median()

#Replacing 0 with median value
df['Year'] = df['Year'].apply(lambda x: x if x!=0 else median_year)

#converting into int
df['Year'] = df['Year'].astype(int)
```

In [13]:
```
# Fuel Type

df['Fuel'].value_counts()
```

Out[13]:
```
Diesel          7168
Petrol          9400
CNG & Hybrids    1345
-                  87
Name: Fuel, dtype: int64
```

In [14]:
```
# mode for fuel
Fuel_mode = df['Fuel'].mode()
Fuel_mode
```

Out[14]:
```
0    Petrol
dtype: object
```

In [15]:
```
# Fuel type
df['Fuel'] = df['Fuel'].apply(lambda x: x if x!='-' else 0)

#Replacing 0 with mode value
df['Fuel'] = df['Fuel'].apply(lambda x: x if x!=0 else "Petrol")

df['Fuel'].value_counts()
```

Out[15]:
```
Petrol          9487
Diesel          7168
CNG & Hybrids    1345
Name: Fuel, dtype: int64
```

In [16]:
```
# Transmission Type
df['Transmission'].value_counts()
```

Out[16]:
```
Manual       14772
Automatic     3141
-               87
Name: Transmission, dtype: int64
```

In [17]:
```
#replacing the "-" with manual
df['Transmission'] = df['Transmission'].apply(lambda x: x if x!='-' else 'Manual')

df['Transmission'].value_counts()
```

Out[17]:
```
Manual       14859
Automatic     3141
Name: Transmission, dtype: int64
```

In [18]:
```
# Brand
df['Brand'].value_counts()
```

Out[18]:
```
Maruti Suzuki    4476
Hyundai          3585
Toyota           2689
```

```
Honda            1345
Chevrolet        1345
Mahindra         1342
Tata              890
Land Rover        450
Renault           449
Audi              448
BMW               448
Ford              446
-                  87
Name: Brand, dtype: int64
```

In [19]:
```
#replacing the "-" with others
df['Brand'] = df['Brand'].apply(lambda x: x if x!='-' else 'others')
df['Brand'].value_counts()
```

Out[19]:
```
Maruti Suzuki    4476
Hyundai          3585
Toyota           2689
Honda            1345
Chevrolet        1345
Mahindra         1342
Tata              890
Land Rover        450
Renault           449
Audi              448
BMW               448
Ford              446
others             87
Name: Brand, dtype: int64
```

In [20]:
```
# Model
df['Model'].value_counts()
```

Out[20]:
```
Elite i20             899
Innova                899
Cruze                 898
City                  898
Innova Crysta         896
Swift                 896
i20                   895
Swift Dzire           892
Nano                  890
Range Rover Evoque    450
Creta                 450
Ciaz                  449
Duster                449
Eeco                  449
Santro Xing           448
Wagon R               448
Ertiga                448
3 Series              448
XUV300                448
A6                    448
Bolero                448
Getz Prime            448
Etios Liva            448
Beat                  447
Zen Estilo            447
Civic                 447
Alto 800              447
Xylo                  446
Ecosport              446
Fortuner              446
EON                   445
-                      87
Name: Model, dtype: int64
```

In [21]:
```
#replacing the "-" with others in Model
df['Model'] = df['Model'].apply(lambda x: x if x!='-' else 'others')
df['Model'].value_counts()
```

Out[21]:
```
Elite i20              899
Innova                 899
Cruze                  898
City                   898
Innova Crysta          896
Swift                  896
i20                    895
Swift Dzire            892
Nano                   890
Range Rover Evoque     450
Creta                  450
Eeco                   449
Duster                 449
Ciaz                   449
Etios Liva             448
Santro Xing            448
Wagon R                448
Ertiga                 448
XUV300                 448
A6                     448
Bolero                 448
Getz Prime             448
3 Series               448
Zen Estilo             447
Beat                   447
Civic                  447
Alto 800               447
Xylo                   446
Ecosport               446
Fortuner               446
EON                    445
others                  87
Name: Model, dtype: int64
```

In [22]:
```
# Location
df['Location'].value_counts()
```

Out[22]:
```
Sector 37, Gurgaon, Haryana                          1345
Pimpri Chinchwad, Pune, Maharashtra                  1343
Moonnamkutty, Kollam, Kerala                          896
Banaswadi Kasturi Nagar, Bengaluru, Karnataka         895
BTM Layout, Bengaluru, Karnataka                      893
Kalkaji, Delhi, Delhi                                 893
Dwarka, Delhi, Delhi                                  892
Udhana Gam, Surat, Gujarat                            450
Greater Kailash 1, Delhi, Delhi                       450
Airoli, Navi Mumbai, Maharashtra                      450
Preet Vihar, Delhi, Delhi                             449
Greater Noida West, Noida, Uttar Pradesh              449
Netaji Subhash Place, Delhi, Delhi                    449
Thane West, Thane, Maharashtra                        449
Varun Niketan, Delhi, Delhi                           449
Indira Nagar, Bengaluru, Karnataka                    449
Tilak Nagar, Delhi, Delhi                             449
Kankurgachhi, Kolkata, West Bengal                    449
Sector 19D, Chandigarh, Chandigarh                    448
Godhra GIDC, Godhra, Gujarat                          448
Nandanam Nandanam, Chennai, Tamil Nadu                448
Sector 8B, Chandigarh, Chandigarh                     448
Patel Nagar 3, Ghaziabad, Uttar Pradesh               448
Geeta Colony, Delhi, Delhi                            448
Umiya Chowk, Rajkot, Gujarat                          448
MG Road, Indore, Madhya Pradesh                       447
Andheri East, Mumbai, Maharashtra                     447
Rajendra Place, Delhi, Delhi                          447
```

```
Ashok Vihar, Delhi, Delhi                          446
Chelamattom, Perumbavoor, Kerala                   446
Kalase Nagar, Mohol, Maharashtra                   445
-                                                   87
Name: Location, dtype: int64
```

In [23]:
```
#replacing the "-" with others in Model
df['Location'] = df['Location'].apply(lambda x: x if x!='-' else 'others')
df['Location'].value_counts()
```

Out[23]:
```
Sector 37, Gurgaon, Haryana                       1345
Pimpri Chinchwad, Pune, Maharashtra               1343
Moonnamkutty, Kollam, Kerala                       896
Banaswadi Kasturi Nagar, Bengaluru, Karnataka      895
Kalkaji, Delhi, Delhi                              893
BTM Layout, Bengaluru, Karnataka                   893
Dwarka, Delhi, Delhi                               892
Greater Kailash 1, Delhi, Delhi                    450
Udhana Gam, Surat, Gujarat                         450
Airoli, Navi Mumbai, Maharashtra                   450
Thane West, Thane, Maharashtra                     449
Varun Niketan, Delhi, Delhi                        449
Indira Nagar, Bengaluru, Karnataka                 449
Kankurgachhi, Kolkata, West Bengal                 449
Preet Vihar, Delhi, Delhi                          449
Netaji Subhash Place, Delhi, Delhi                 449
Tilak Nagar, Delhi, Delhi                          449
Greater Noida West, Noida, Uttar Pradesh           449
Patel Nagar 3, Ghaziabad, Uttar Pradesh            448
Godhra GIDC, Godhra, Gujarat                       448
Nandanam Nandanam, Chennai, Tamil Nadu             448
Sector 19D, Chandigarh, Chandigarh                 448
Umiya Chowk, Rajkot, Gujarat                       448
Sector 8B, Chandigarh, Chandigarh                  448
Geeta Colony, Delhi, Delhi                         448
MG Road, Indore, Madhya Pradesh                    447
Rajendra Place, Delhi, Delhi                       447
Andheri East, Mumbai, Maharashtra                  447
Ashok Vihar, Delhi, Delhi                          446
Chelamattom, Perumbavoor, Kerala                   446
Kalase Nagar, Mohol, Maharashtra                   445
others                                              87
Name: Location, dtype: int64
```

In [24]:
```
# Location
df['Variant'].value_counts()
```

Out[24]:
```
-                                  1431
2.8Z Automatic                      896
Petrol HSE Dynamic                  450
2.5 G (Diesel) 8 Seater             450
Sportz (O) 1.4                      450
1.6 SX Plus Auto                    450
Smart Hybrid Alpha                  449
i-VTEC S                            449
1.2 Vxi BSIV                        449
i-VTEC VX                           449
LTZ                                 449
2004-2011 2.5 G4 Diesel 7-seater    449
2012-2015 85PS Diesel RxL           449
5 STR With AC Plus HTR CNG          449
2012 LTZ AT                         449
Asta 1.2 (O)                        449
VXI CNG                             448
GLS                                 448
2011-2014 VDI                       448
1.1 GVS Option                      448
SLX                                 448
1.2 VX                              448
```

```
2012-2014 Magna Optional 1.2         448
2006-2010 LXI Minor                  448
320d Sport                           448
2012-2016 VXT                        447
LXI BS IV                            447
V                                    447
Diesel LS                            447
1.2 Magna Executive                  447
1.5 TDCi Titanium Plus               446
2011-2016 4x2 Manual                 446
2009-2011 Lx BSIV                    446
2009-2011 E4 BS IV                   446
Era +                                445
2009-2011 Cx BSIV                    444
LXI Option                           443
Name: Variant, dtype: int64
```

In [25]:

```
#replacing the "-" with others in Model
df['Variant'] = df['Variant'].apply(lambda x: x if x!='-' else 'others')
df['Variant'].value_counts()
```

Out[25]:

```
others                               1431
2.8Z Automatic                        896
2.5 G (Diesel) 8 Seater               450
Sportz (O) 1.4                        450
1.6 SX Plus Auto                      450
Petrol HSE Dynamic                    450
2004-2011 2.5 G4 Diesel 7-seater      449
i-VTEC S                              449
1.2 Vxi BSIV                          449
i-VTEC VX                             449
LTZ                                   449
Smart Hybrid Alpha                    449
Asta 1.2 (O)                          449
2012-2015 85PS Diesel RxL             449
2012 LTZ AT                           449
5 STR With AC Plus HTR CNG            449
GLS                                   448
2012-2014 Magna Optional 1.2          448
1.2 VX                                448
320d Sport                            448
VXI CNG                               448
SLX                                   448
2006-2010 LXI Minor                   448
2011-2014 VDI                         448
1.1 GVS Option                        448
1.2 Magna Executive                   447
V                                     447
LXI BS IV                             447
2012-2016 VXI                         447
Diesel LS                             447
1.5 TDCi Titanium Plus                446
2011-2016 4x2 Manual                  446
2009-2011 E4 BS IV                    446
2009-2011 Lx BSIV                     446
Era +                                 445
2009-2011 Cx BSIV                     444
LXI Option                            443
Name: Variant, dtype: int64
```

In [26]:

```
df.dtypes
```

Out[26]:

```
Brand             object
Model             object
Variant           object
Year               int32
Selling_Price    float64
KM_Driven        float64
Fuel              object
```

```
Transmission      object
Location          object
dtype: object
```

In [27]:
```
df.head(10)
```

Out[27]:

	Brand	Model	Variant	Year	Selling_Price	KM_Driven	Fuel	Transmission	Location
0	Hyundai	i20	2014 Magna Optional 1.2	2013	3.400000e+05	63000.000000	Petrol	Manual	Geeta Color Delhi, Del Nandana
1	Audi	A6	others	2016	2.650000e+06	72000.000000	Petrol	Automatic	Nandana Chennai, Tan
2	Maruti Suzuki	Alto 800	2012-2016 VXI	2016	2.950000e+05	37739.000000	Petrol	Manual	Andheri Ea Mumb Maharasht
3	Maruti Suzuki	Swift	others	2007	2.850000e+05	38000.000000	Petrol	Manual	Kasturi Naga Bengalu Karnata
4	Maruti Suzuki	Swift Dzire	1.2 Vxi BSIV	2015	4.700000e+05	51000.000000	Petrol	Manual	Preet Vih Delhi, Del
5	Chevrolet	Beat	Diesel LS	2013	2.850000e+05	43000.000000	Diesel	Manual	Kasturi Naga Bengalu Karnata
6	Maruti Suzuki	Wagon R	2006-2010 LXI Minor	2008	1.750000e+05	47000.000000	Petrol	Manual	Moonnamkut Kollam, Ker
7	Chevrolet	Cruze		2013	3.250000e+05	73000.000000	Diesel	Manual	Delhi, Del
8	others	others	others	2013	6.628532e+05	65147.551278	Petrol	Manual	othe
9	Hyundai	Elite i20	Asta 1.2 (O)	2019	8.450000e+05	26000.000000	Petrol	Manual	Indira Naga Bengalu Karnata

In [28]:
```
#checking all the column names
list(df.columns)
```

Out[28]:
```
['Brand',
 'Model',
 'Variant',
 'Year',
 'Selling_Price',
 'KM_Driven',
 'Fuel',
 'Transmission',
 'Location']
```

In [29]:
```
df = df[['Brand', 'Variant', 'Model', 'Year', 'Transmission', 'Fuel', 'KM_Driven', '
df.sample(5)
```

Out[29]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Location	Selling_P
15237	Hyundai	1.1 GVS Option	Getz Prime	2009	Manual	Petrol	38000.0	Pimpri Chinchwad, Pune, Maharashtra	1600
16231	Mahindra	others	XUV300	2019	Manual	Diesel	19000.0	Sector 8B, Chandigarh, Chandigarh	10500
11373	Maruti Suzuki	LXI Option	Swift Dzire	2015	Manual	Petrol	48000.0	Dwarka, Delhi, Delhi	3950
9883	Maruti Suzuki	others	Swift	2007	Manual	Petrol	38000.0	Banaswadi Kasturi Nagar, Bengaluru, Karnataka	2850
4726	Maruti Suzuki	2006-2010 LXI Minor	Wagon R	2008	Manual	Petrol	47000.0	Moonnamkutty, Kollam, Kerala	1950

In [30]:
```
(df['Brand'] == 'others').sum()
```

Out[30]:
```
87
```

In [31]:
```
df.shape
```

Out[31]:
```
(18000, 9)
```

In [32]:
```
drop87 = df[df['Brand'] == 'others'].index
df.drop(drop87, inplace = True)
```

In [33]:
```
df.shape
```

Out[33]:
```
(17913, 9)
```

In [34]:
```
(df['Brand'] == 'others').sum()
```

Out[34]:
```
0
```

In [35]:
```
#checking the data type and null values of the variables in the data-set
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17913 entries, 0 to 17999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----

 0   Brand          17913 non-null  object
 1   Variant        17913 non-null  object
 2   Model          17913 non-null  object
 3   Year           17913 non-null  int32
 5   Fuel           17913 non-null  object
 6   KM_Driven      17913 non-null  float64
 7   Location       17913 non-null  object
 8   Selling_Price  17913 non-null  float64
dtypes: float64(2), int32(1), object(6)
memory usage: 1.3+ MB
```

In [36]:
```
df.describe().T
```

Out[36]:

	count	mean	std	min	25%	50%	75%	ma
Year	17913.0	2013.352426	3.624825	2006.0	2010.0	2013.0	2016.0	2020.
KM_Driven	17913.0	65463.960420	33416.204594	14000.0	38000.0	62000.0	78001.0	140000.
Selling_Price	17913.0	664286.851672	761820.570473	55000.0	295000.0	360000.0	774000.0	4750000.

In [37]:
```
# Describing object types
df.describe(include='object').T
```

Out[37]:

	count	unique	top	freq
Brand	17913	12	Maruti Suzuki	4476
Variant	17913	37	others	1344
Model	17913	31	Elite i20	899
Transmission	17913	2	Manual	14772
Fuel	17913	3	Petrol	9400
Location	17913	31	Sector 37, Gurgaon, Haryana	1345

Handling the Null Values :

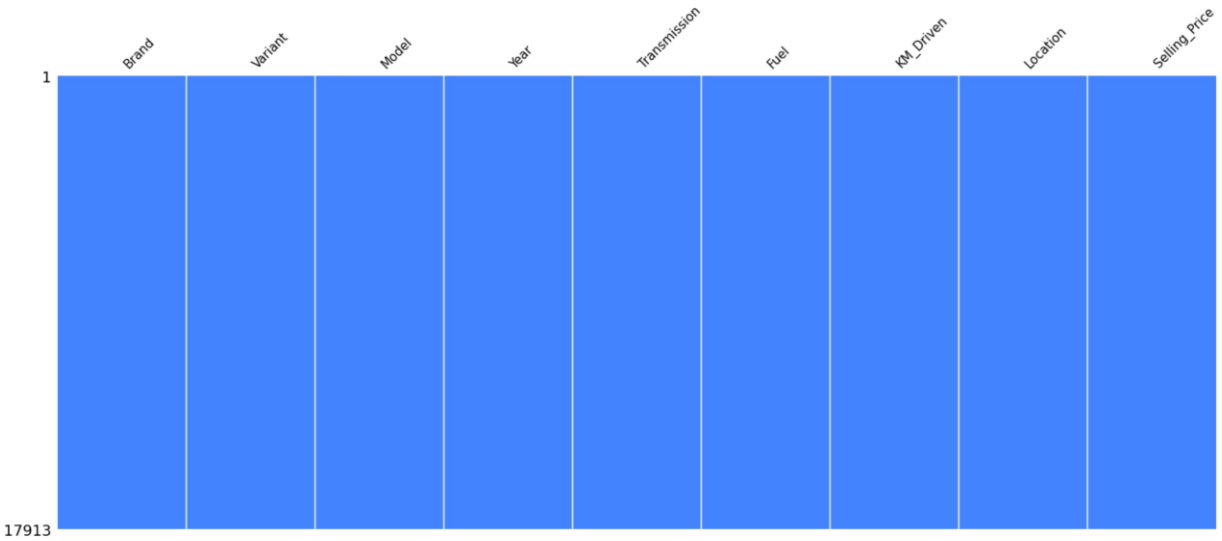
In [38]:
```
#checking the null values
for col in df.columns:print("\nTitle :",col,"\nNaN val:",df[col].isnull().sum())
```

```
Title : Brand
NaN val: 0

Title : Variant
NaN val: 0

Title : Model
NaN val: 0

Title : Year
NaN val: 0

Title : Transmission
NaN val: 0

Title : Fuel
NaN val: 0

Title : KM_Driven
NaN val: 0

Title : Location
NaN val: 0

Title : Selling_Price
NaN val: 0
```

In [39]:
```
# Program to visualize missing values in dataset
# Importing the libraries
import missingno as msno
```

```
# Visualize missing values as a matrix
msno.matrix(df,labels=True, sparkline=False, figsize=(25,10), fontsize=15, color=(0.
```

Out[39]: <AxesSubplot:>



Catagorical Variables :

In [40]:
```
cat_List = [x for x in df.columns if df[x].dtype==object]
list (cat_List)
```

Out[40]: ['Brand', 'Variant', 'Model', 'Transmission', 'Fuel', 'Location']

Continous Variables :

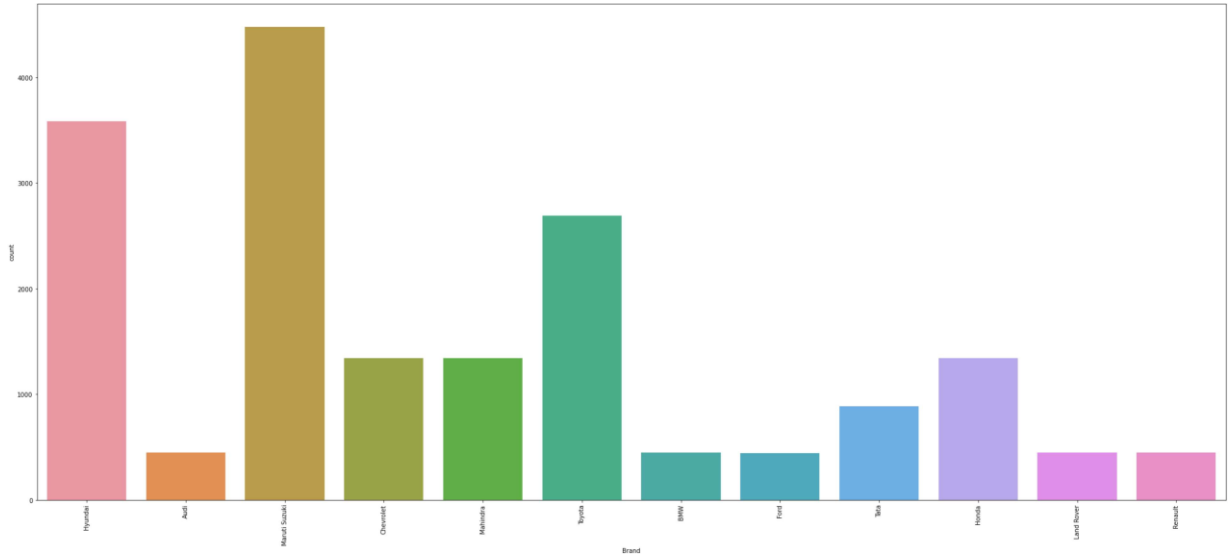
In [41]:
```
num_List = [x for x in df.columns if x not in cat_List]
list (num_List)
```

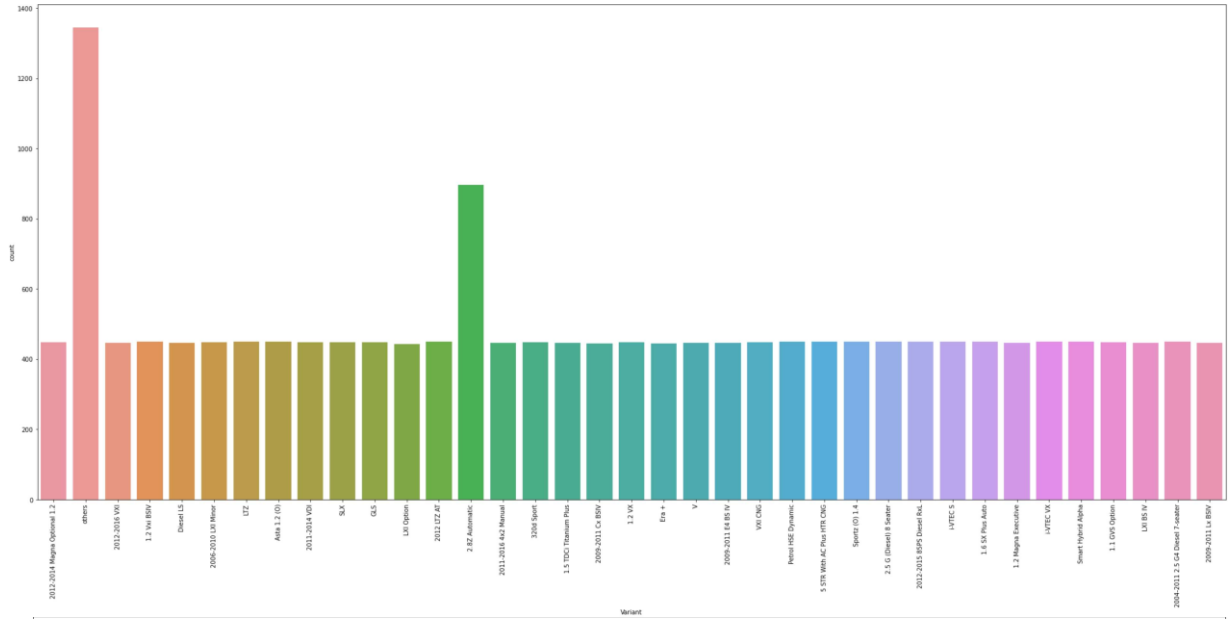
Out[41]: ['Year', 'KM_Driven', 'Selling_Price']

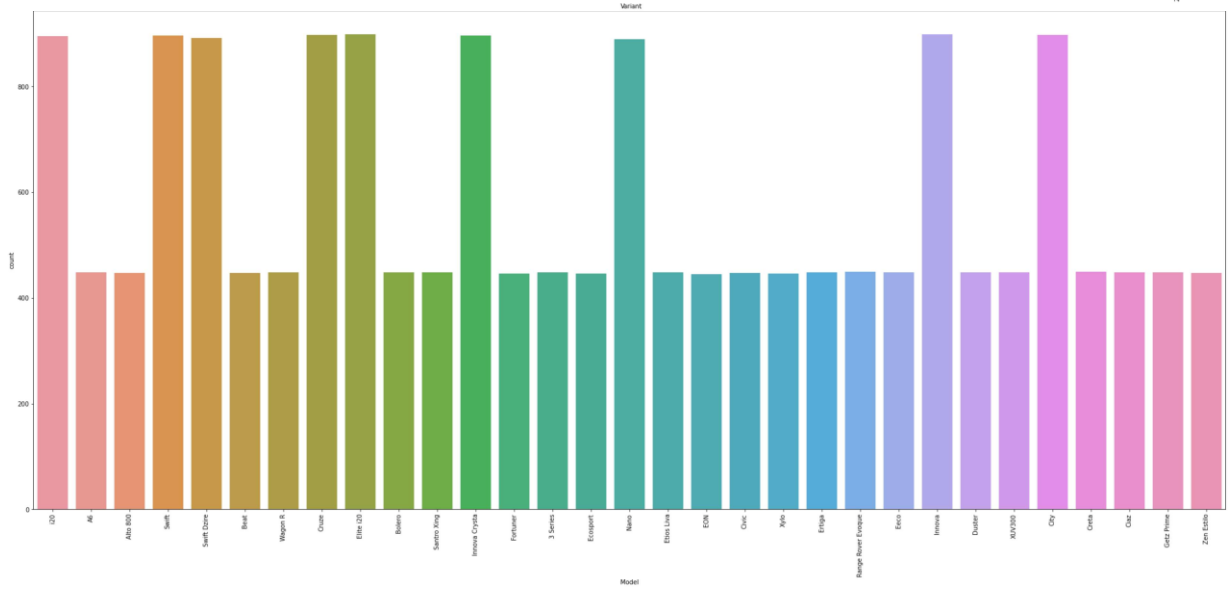
DATA VISUALIZATION :

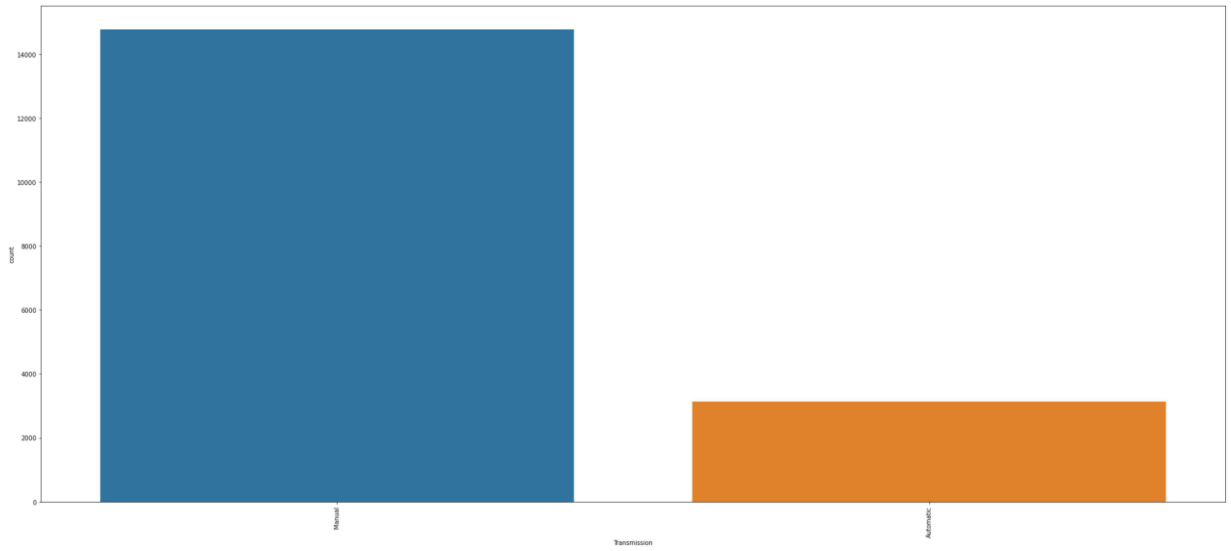
UNI-VARIATE ANALYSIS :

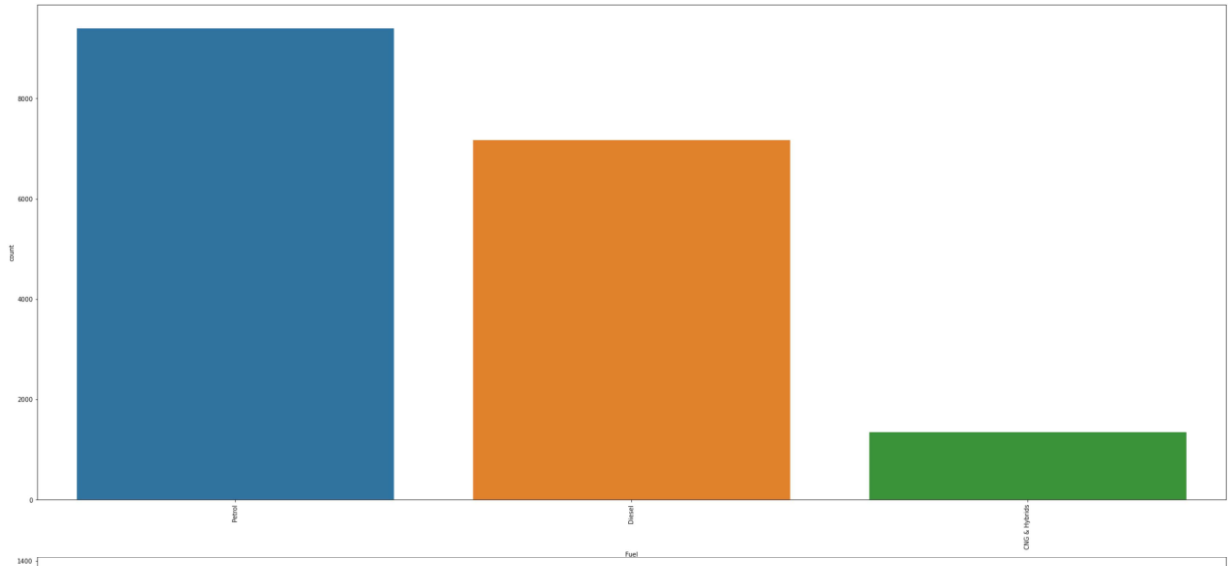
In [42]:
```
for i in cat_List:
    plt.figure(figsize=(35,15))
    sn.countplot(df[i])
    plt.xticks(rotation=90)
    plt.show()
```

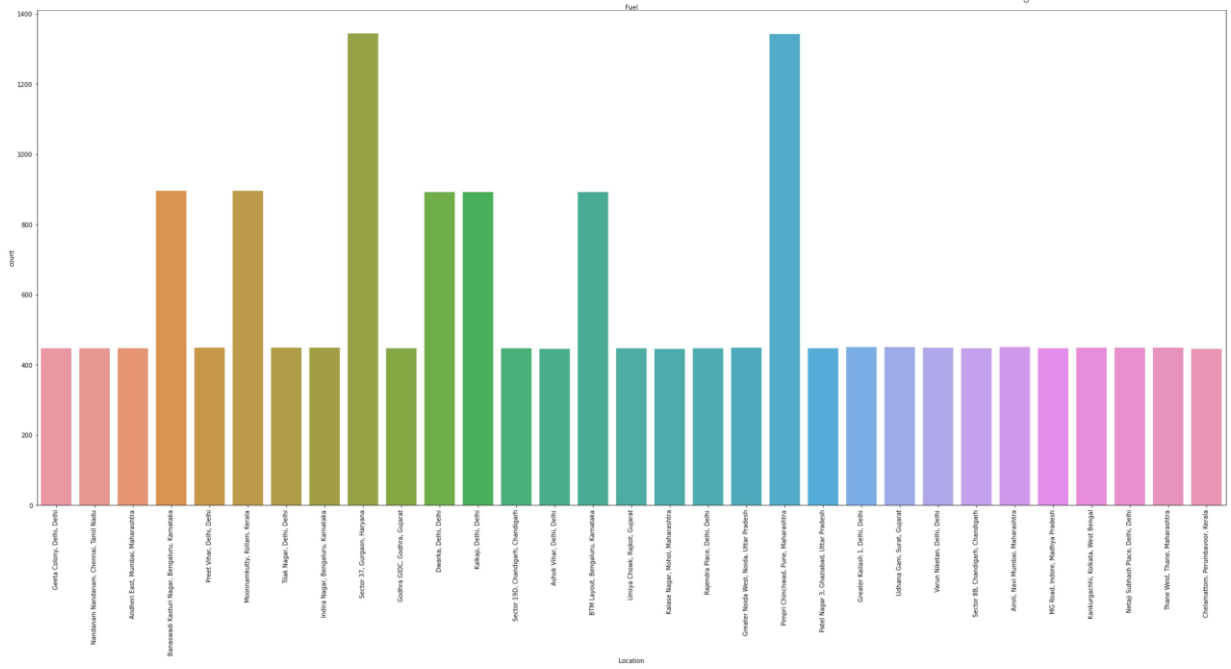

count
Brand
Hyundai
Audi
Maruti Suzuki
Chevrolet
Mahindra
Toyota
BMW
Ford
Tata
Honda
Land Rover
Renault


count
Variant
others
2012-2014 Magna Optional 1.2
2012-2016 VXi
1.2 Vxi BSIV
Diesel LS
2006-2010 LXi Minor
LTZ
Asta 1.2 (O)
2011-2014 VDi
SLX
GLS
LXi Option
2012 DZ AT
2 BZ Automatic
2011-2014 4x2 Manual
320d Sport
1.5 TDCI Titanium Plus
2009-2011 Ce BSIV
1.2 VX
Era +
V
2009-2011 E4 BS IV
VXi CNG
Petrol HSE Dynamic
5 STR With AC Plus HTR CNG
Sportz (O) 1.4
2.5 G (Diesel) 8 Seater
2012-2015 RXPS Diesel Rxl
i-VTEC S
1.6 SX Plus Auto
1.2 Magna Executive
i-VTEC VX
Smart Hybrid Alpha
1.1 GVS Option
LXi BS IV
2004-2011 2.5 G4 Diesel 7-seater
2009-2011 Lx BSIV


count
Model
i20
Alt
Alto 800
Swift
Swift Dzire
Ikwt
Wagon R
Cruze
Elite i20
Bolero
Santro Xing
Innova Crysta
Fortuner
3 Series
Ecosport
Nano
Etios Liva
EON
Civic
Aylo
Enjoy
Range Rover Evoque
Enco
Innova
Duster
XUV300
City
Creta
Cez
Getz Prime
Zen Estilo





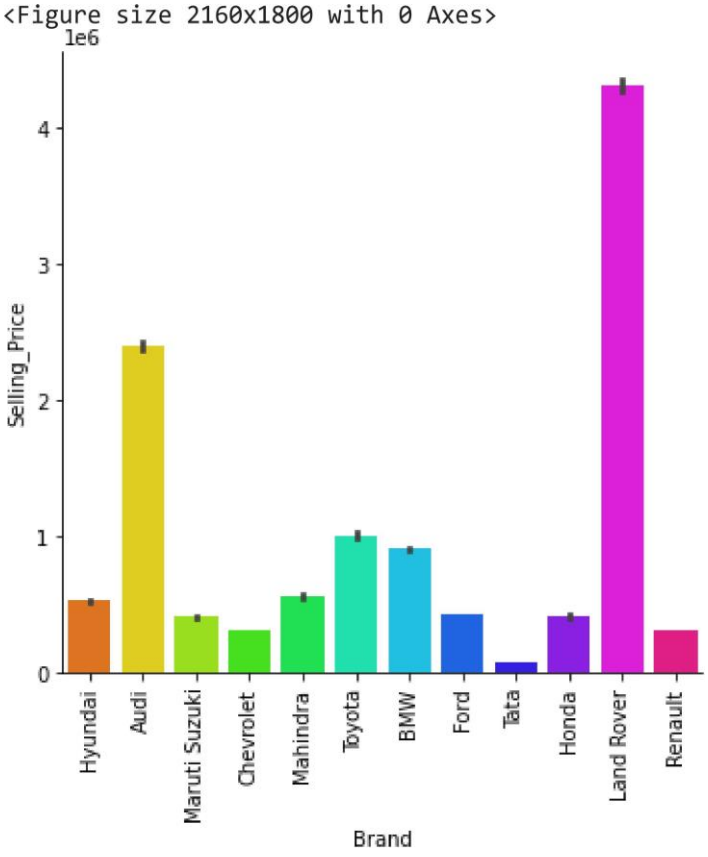


BI-VARIATE ANALYSIS :

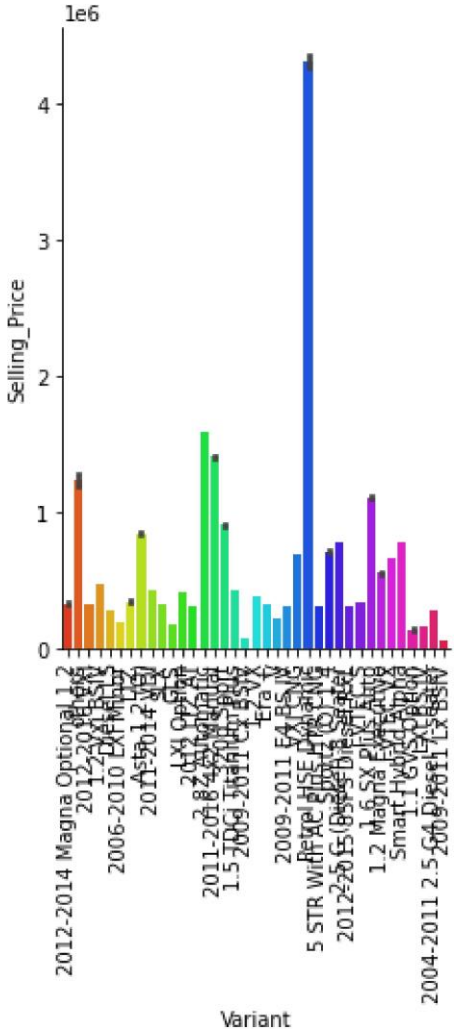
In [43]:
```
for i in cat_List:
    plt.figure(figsize=(30,25))
    sn.catplot(y='Selling_Price',x=i,data=df,kind="bar",palette="hsv")
```

```
    plt.xticks(rotation=90)
    plt.show()
```

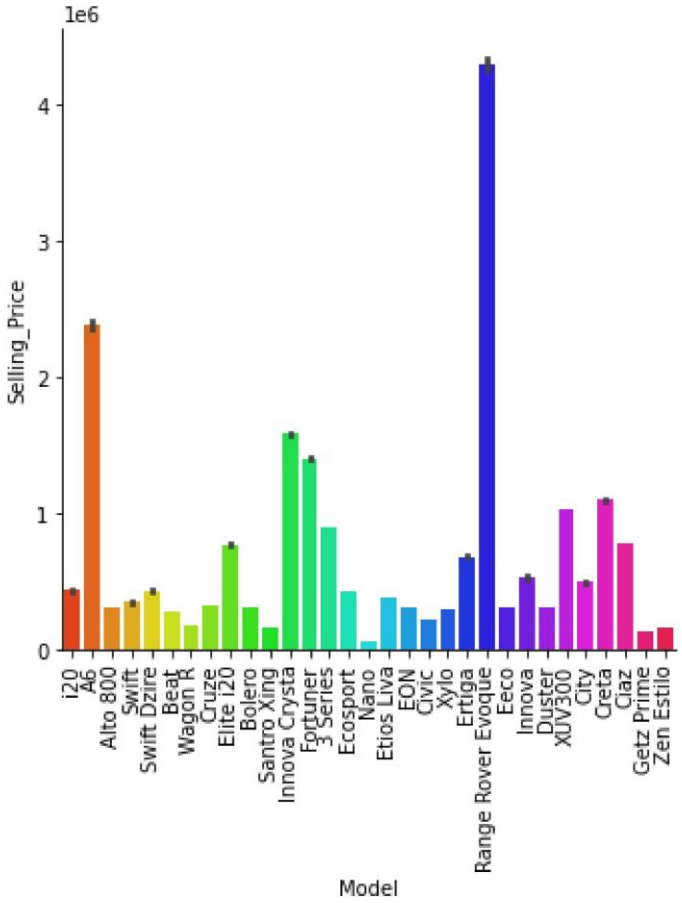
```
<Figure size 2160x1800 with 0 Axes>
```



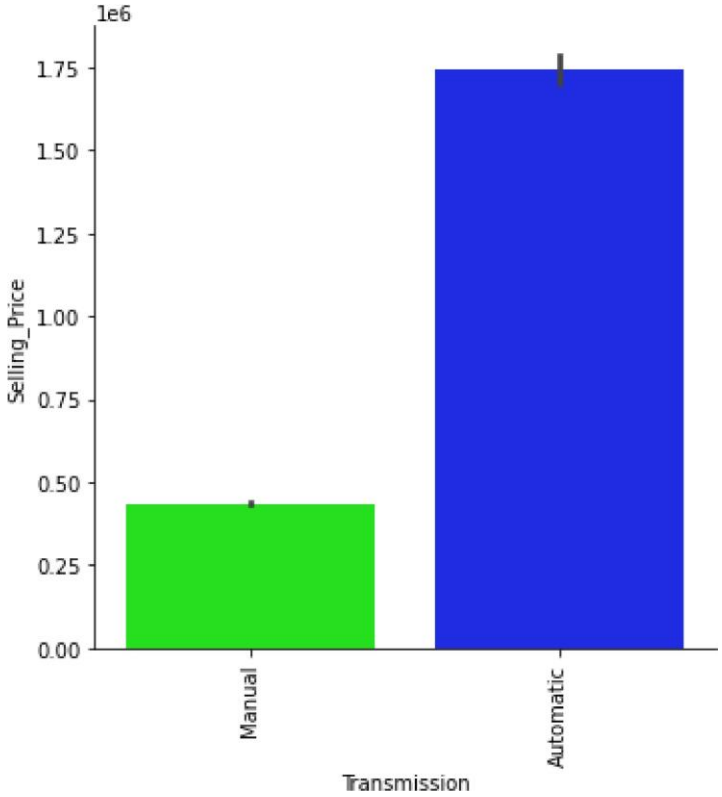

```
<Figure size 2160x1800 with 0 Axes>
```



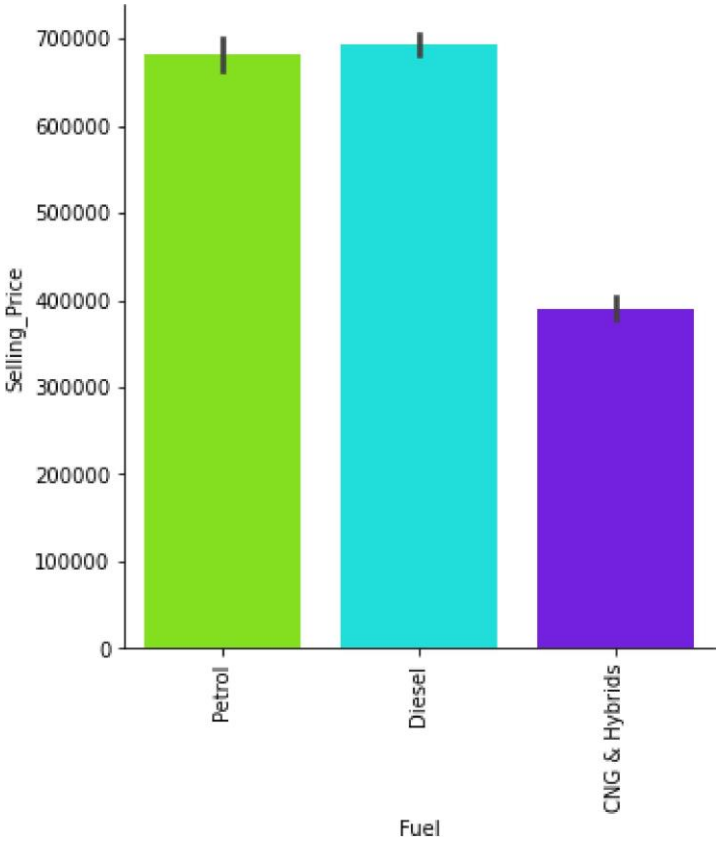
<Figure size 2160x1800 with 0 Axes>



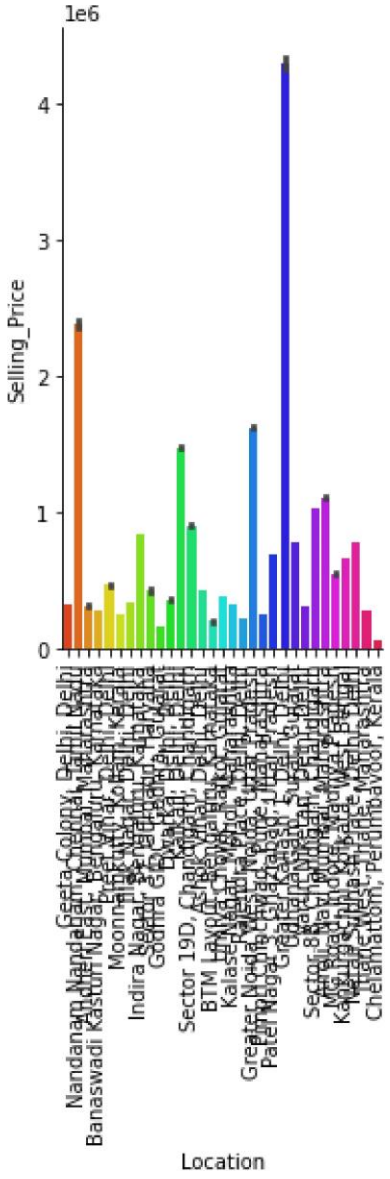
<Figure size 2160x1800 with 0 Axes>



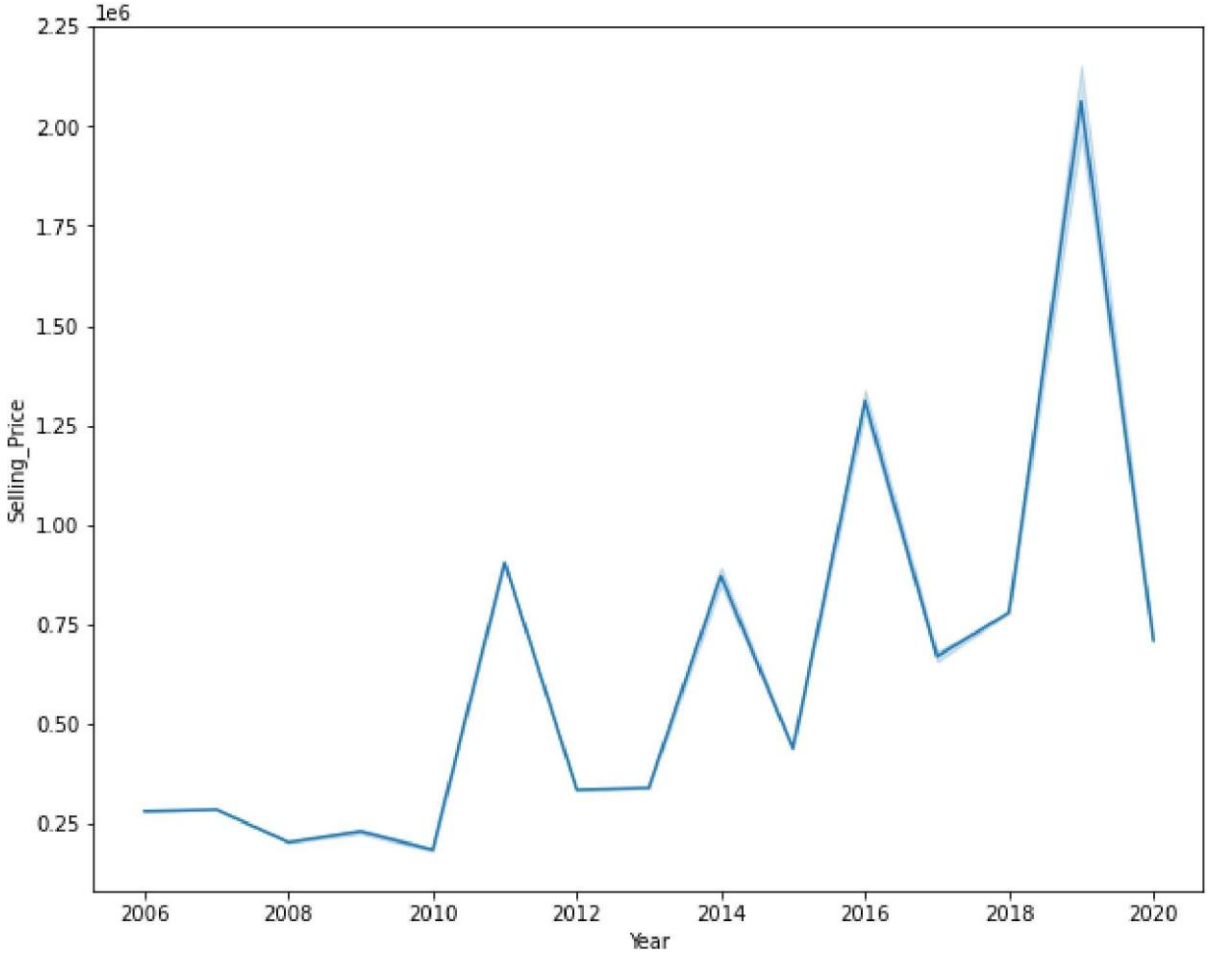
<Figure size 2160x1800 with 0 Axes>

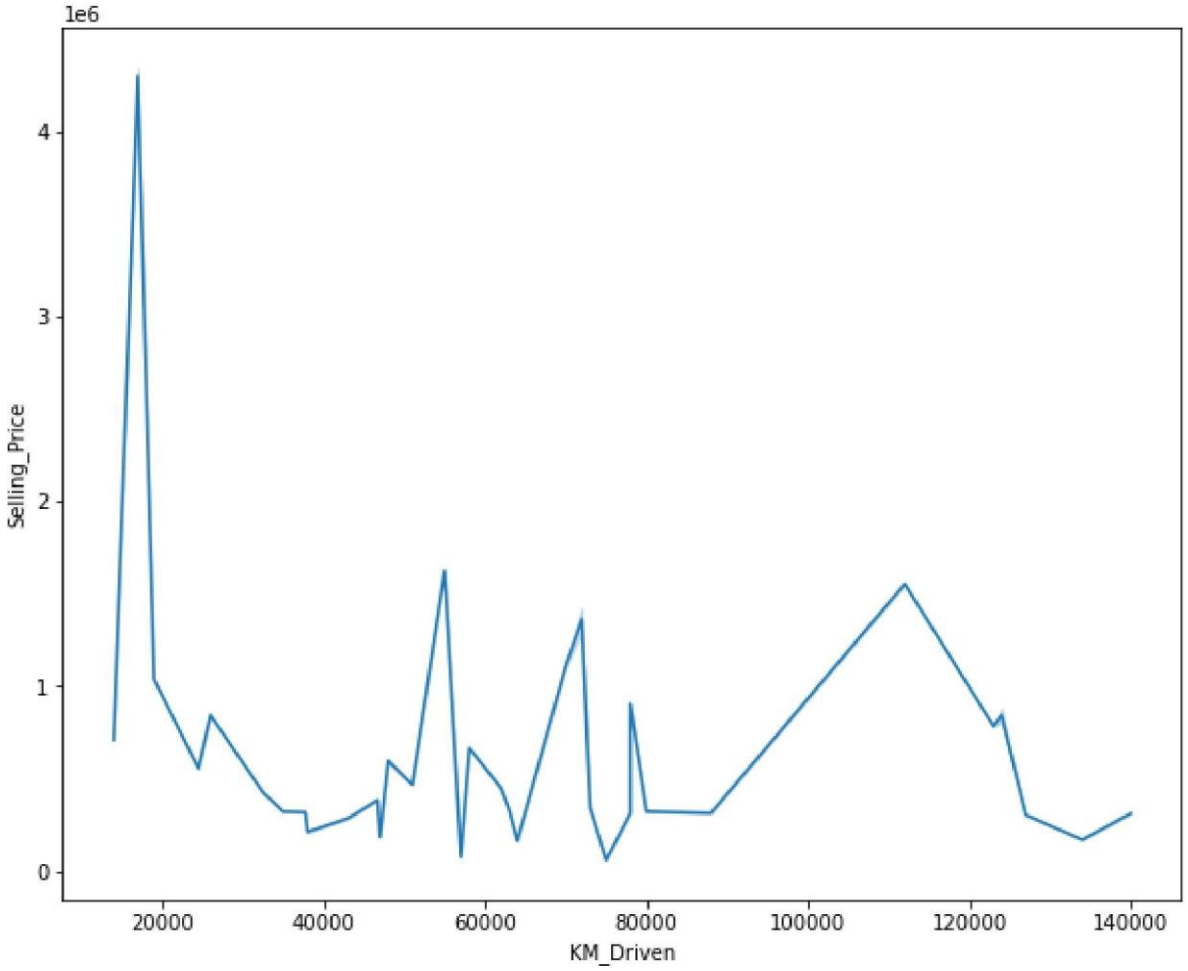


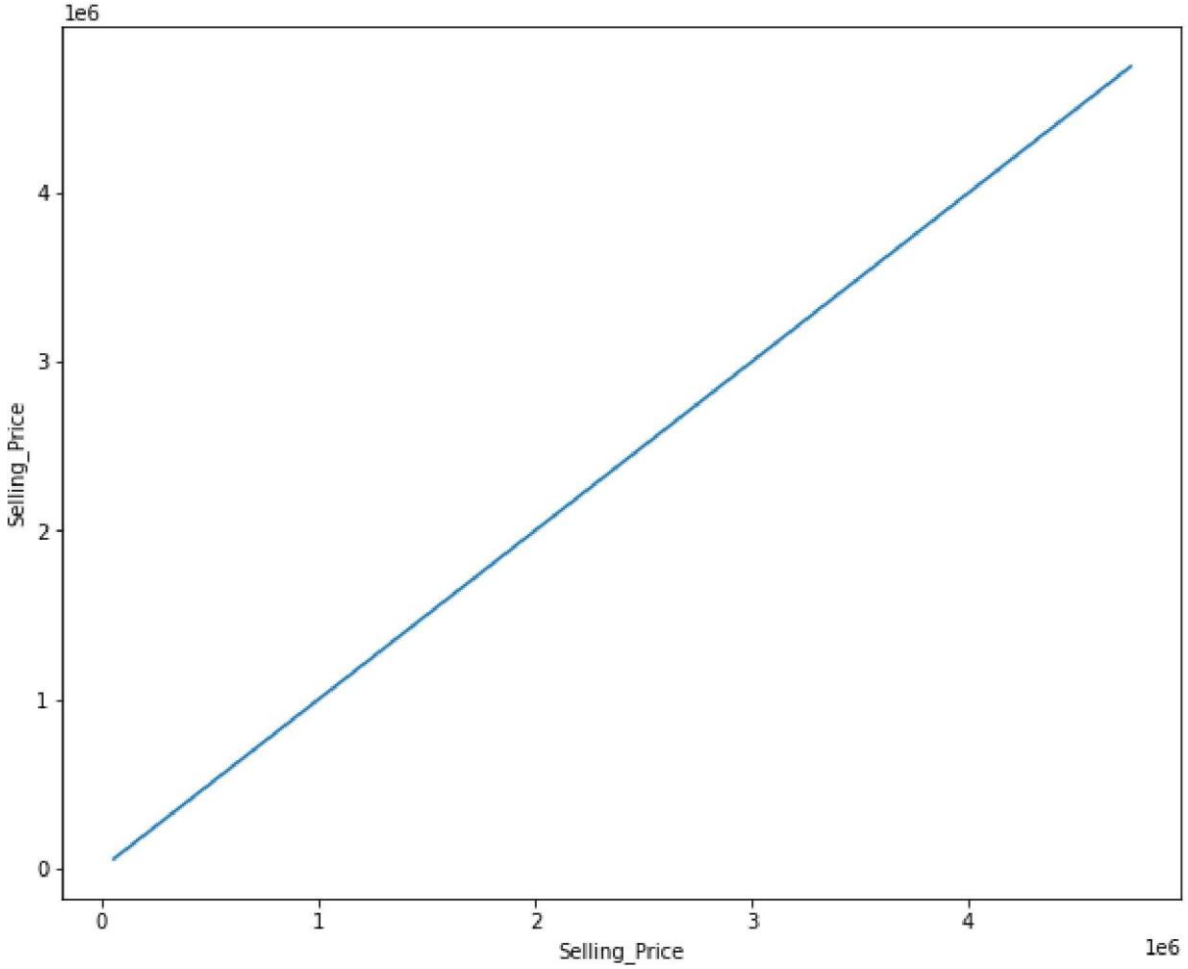

```
<Figure size 2160x1800 with 0 Axes>
```



In [44]:
```
for i in num_List:
    plt.figure(figsize=(10,8))
    sn.lineplot(y='Selling_Price',x=i,data=df)
    plt.show()
```


1e6
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Selling_Price
2006
2008
2010
2012
2014
2016
2018
2020
Year


1e6
4
3
2
1
0
Selling_Price
20000
40000
60000
80000
100000
120000
140000
KM_Driven

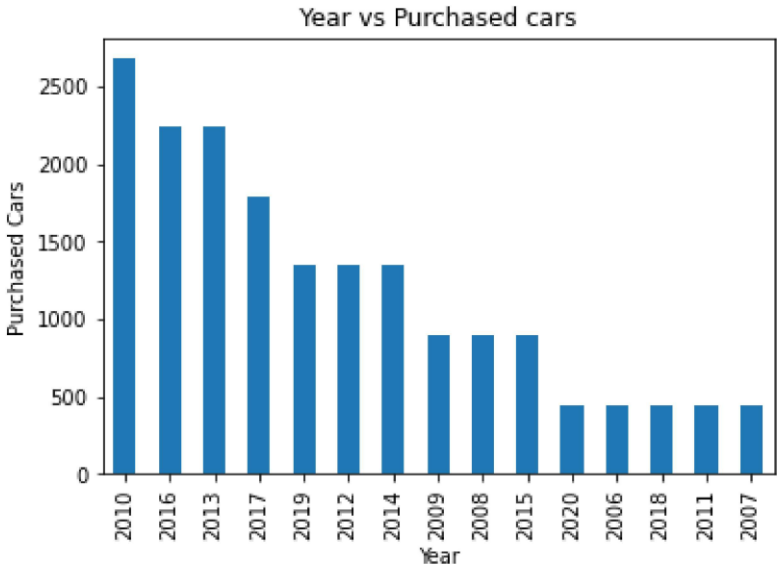


In [45]:
```
df.head(0)
```

Out[45]:

Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Location	Selling_Price

In [46]:
```
#Plotting year vs no of cars
purchased_car_per_year = df['Year'].value_counts()
purchased_car_per_year.plot(kind='bar')
plt.xlabel("Year")
plt.ylabel("Purchased Cars")
plt.title("Year vs Purchased cars")
plt.show()
```



Feature Engineering:

Now we will use Label Encoder to change catagorical values to Numerical values.

LabelEncoder

In [47]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in cat_List:
    df[i] = le.fit_transform(df[i].astype(str))
print (df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17913 entries, 0 to 17999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Brand          17913 non-null  int32  
 1   Variant        17913 non-null  int32  
 2   Model          17913 non-null  int32  
 3   Year           17913 non-null  int32  
 4   Transmission   17913 non-null  int32  
 5   Fuel           17913 non-null  int32  
 6   KM_Driven      17913 non-null  float64
 7   Location       17913 non-null  int32  
 8   Selling_Price  17913 non-null  float64
dtypes: float64(2), int32(7)
memory usage: 1.5 MB
None
```

In [48]:
```
#Checking the dataset
df.head()
```

Out[48]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Location	Selling_Price
0	5	16	30	2013	1	2	63000.0	7	340000.0
1	0	36	1	2016	0	2	72000.0	17	2650000.0
2	8	18	2	2016	1	2	37739.0	1	295000.0
3	8	36	24	2007	1	2	38000.0	4	285000.0
4	8	3	25	2015	1	2	51000.0	21	470000.0

In [49]:
```
#Statistical summary of the dataset
df.describe()
```

Out[49]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_
count	17913.000000	17913.000000	17913.000000	17913.000000	17913.000000	17913.000000	17913.
mean	6.598727	18.627980	15.818512	2013.352426	0.824652	1.449673	65463.
std	2.990211	11.151508	8.736219	3.624825	0.380275	0.630603	33416.
min	0.000000	0.000000	0.000000	2006.000000	0.000000	0.000000	14000.
25%	5.000000	8.000000	8.000000	2010.000000	1.000000	1.000000	38000.

	Brand	Variant	Model	Year	Transmission	Fuel	KM_
50%	7.000000	19.000000	16.000000	2013.000000	1.000000	2.000000	62000.(
75%	8.000000	29.000000	23.000000	2016.000000	1.000000	2.000000	78001.(
max	11.000000	36.000000	30.000000	2020.000000	1.000000	2.000000	140000.(

In [50]:
```
#Checking correlation of the dataset
corr=df.corr()  #corr() function provides the correlation value of each column
corr
```

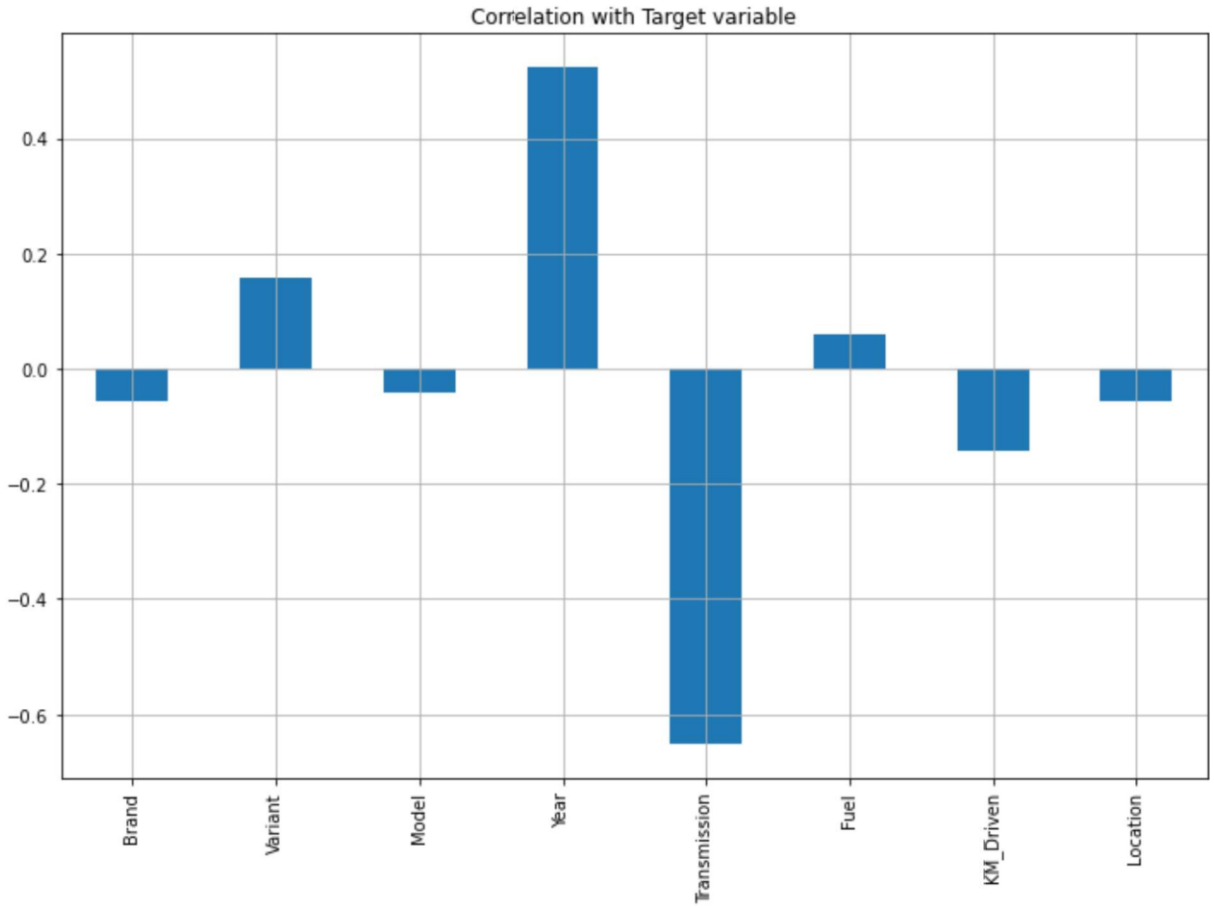
Out[50]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Locati
Brand	1.000000	-0.357354	0.474785	-0.097842	0.224600	-0.104210	0.246958	0.174
Variant	-0.357354	1.000000	-0.319339	0.154980	0.079168	-0.025789	-0.198442	-0.015
Model	0.474785	-0.319339	1.000000	-0.084001	0.231520	0.091163	-0.048848	0.081
Year	-0.097842	0.154980	-0.084001	1.000000	-0.246251	0.041037	-0.429918	-0.041
Transmission	0.224600	0.079168	0.231520	-0.246251	1.000000	0.014999	-0.046171	0.198
Fuel	-0.104210	-0.025789	0.091163	0.041037	0.014999	1.000000	-0.529789	-0.235
KM_Driven	0.246958	-0.198442	-0.048848	-0.429918	-0.046171	-0.529789	1.000000	0.182
Location	0.174485	-0.015510	0.081709	-0.041495	0.198406	-0.235309	0.182995	1.000(
Selling_Price	-0.055908	0.158498	-0.042482	0.523487	-0.652780	0.060960	-0.142475	-0.057

In [51]:
```
#Plotting heatmap for visualizing the correlation
plt.figure(figsize=(15,12))
sn.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',annot=True)
plt.show()
```

In [52]:
```
#Correlation with target variable
plt.figure(figsize=(12,8))
df.drop('Selling_Price',axis=1).corrwith(df['Selling_Price']).plot(kind='bar',grid=T
plt.title('Correlation with Target variable')
```

Out[52]: Text(0.5, 1.0, 'Correlation with Target variable')

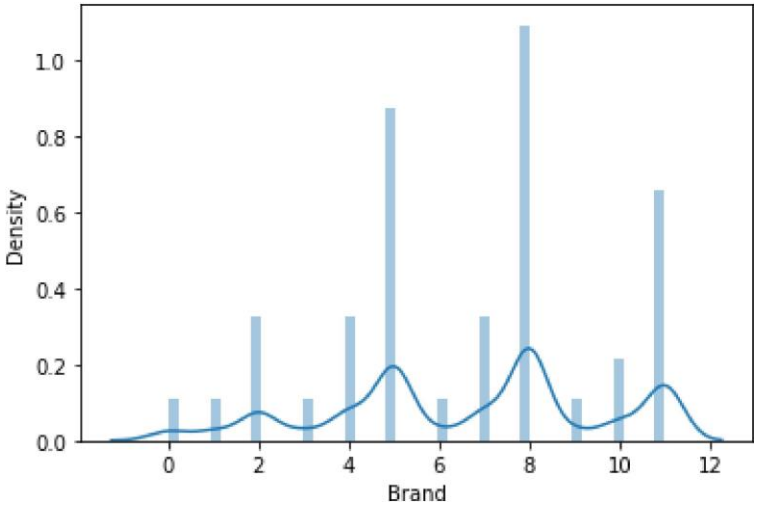


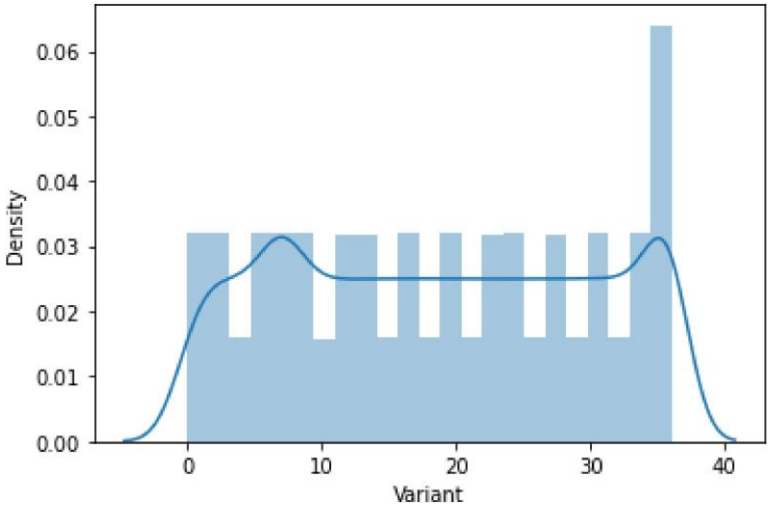
Checking skewness :

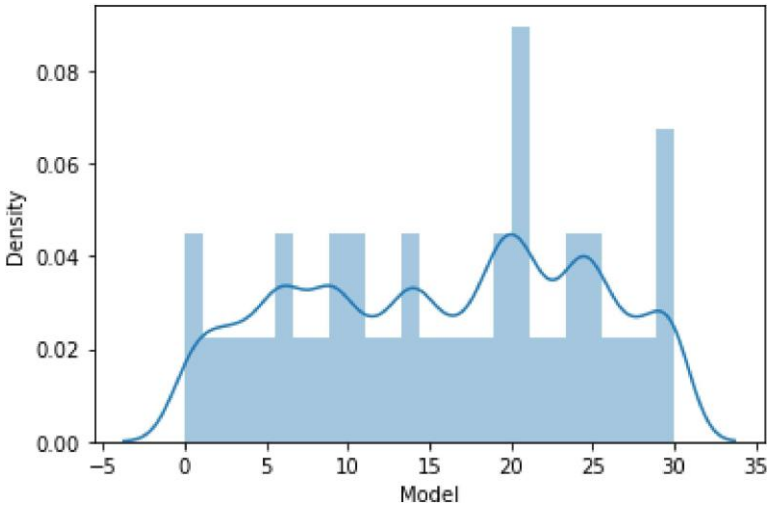
In [53]:
```
#checking the skewness
df.skew()
```

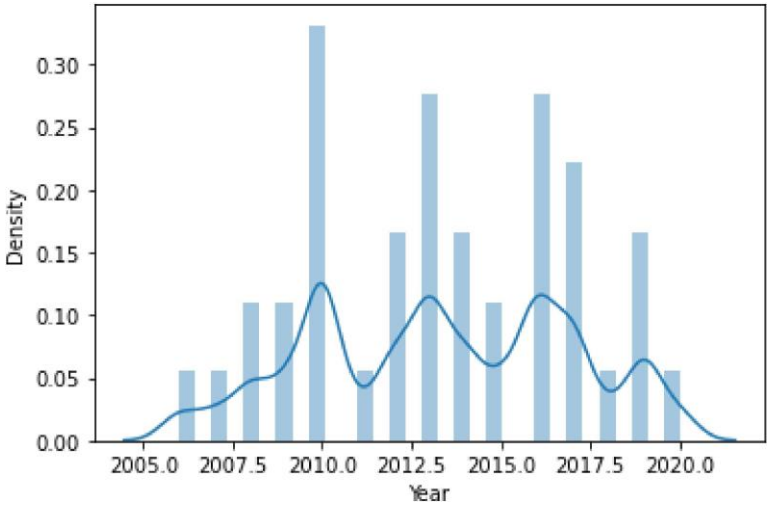
Out[53]:
```
Brand           -0.238892
Variant          0.017713
Model           -0.111916
Year            -0.072629
Transmission    -1.707654
Fuel            -0.708647
KM_Driven        0.692775
Location        -0.037121
Selling_Price    3.237458
dtype: float64
```

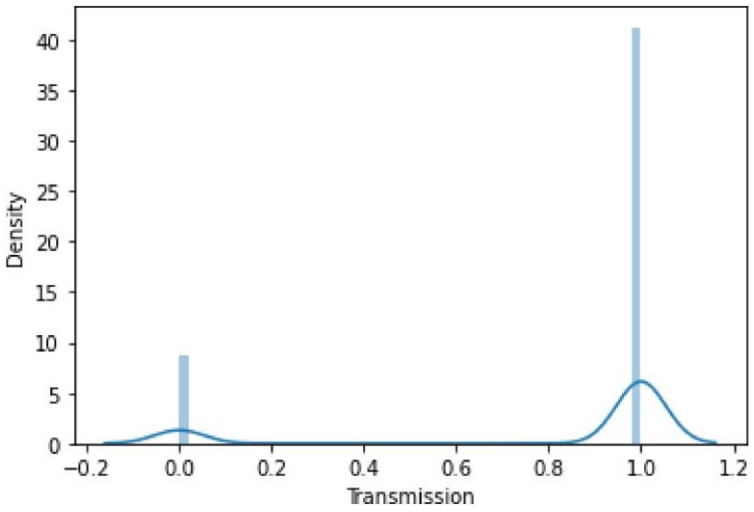
In [54]:
```
#Plotting distplot for checking the distribution of skewness
for col in df.describe().columns:
    sn.distplot(df[col])
    plt.show()
```

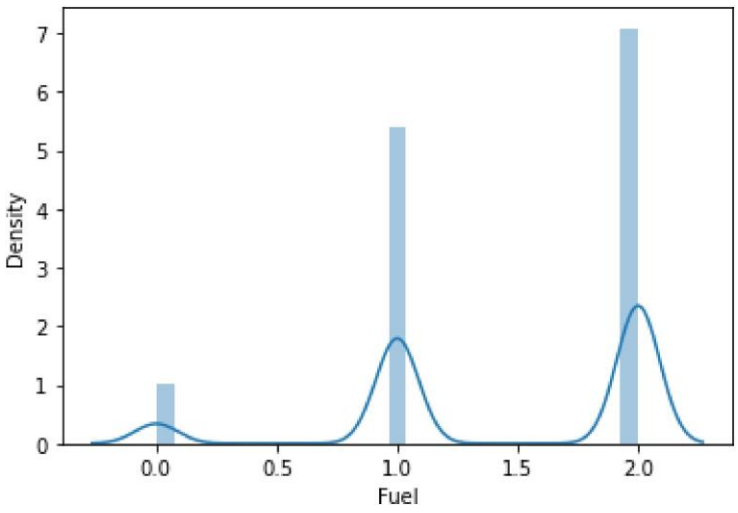

Density
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
Brand

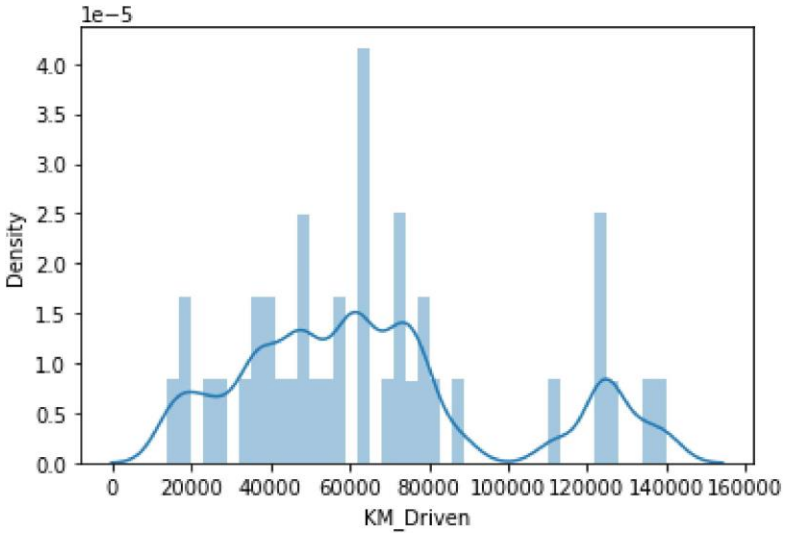

Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
Variant

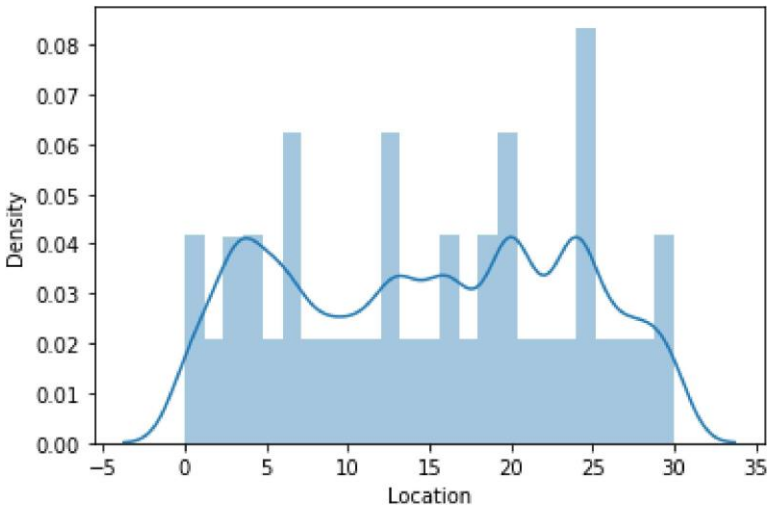

Density
0.08
0.06
0.04
0.02
0.00
−5
0
5
10
15
20
25
30
35
Model

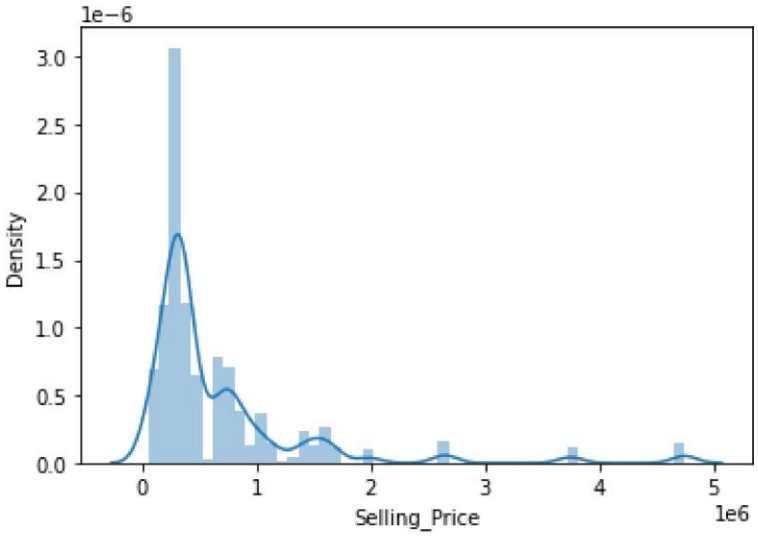

Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5
2020.0
Year


Density
40
35
30
25
20
15
10
5
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Transmission


Density
7
6
5
4
3
2
1
0
0.0
0.5
1.0
1.5
2.0
Fuel

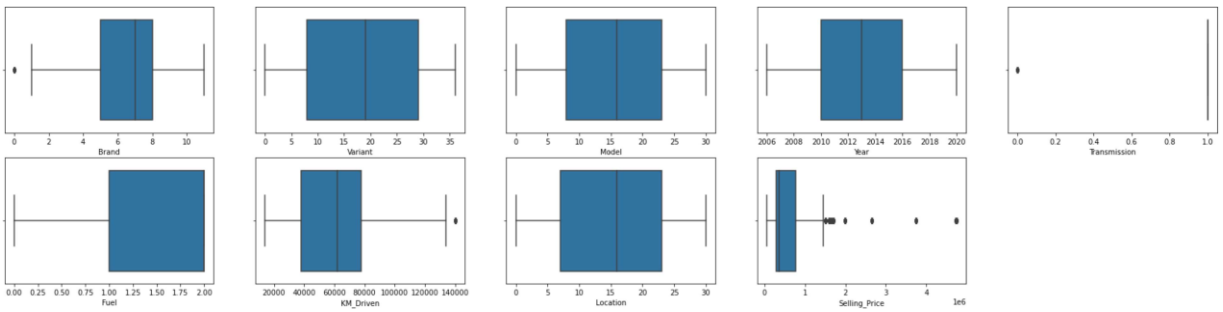






Checking outliers :

In [55]:
```
collist=df.columns.values
ncol=5
nrow=8
plt.figure(figsize=(30,30))
for i in range(0,len(collist)):
    plt.subplot(nrow,ncol,i+1)
    sn.boxplot(df[collist[i]])
```



Handling outliers by using z-score method :

In [56]:
```
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(df))   #converting all values into absolute values
threshold=3    #setting up a threshold
np.where(z>3)
```

Out[56]:
```
(array([   25,    65,   105,   145,   185,   225,   265,   305,   345,
          385,   425,   464,   504,   544,   584,   624,   664,   704,
          744,   784,   824,   864,   904,   944,   984,  1024,  1064,
         1104,  1144,  1184,  1224,  1264,  1304,  1344,  1384,  1424,
         1463,  1502,  1542,  1582,  1622,  1662,  1701,  1741,  1781,
         1821,  1861,  1900,  1940,  1980,  2020,  2060,  2100,  2140,
         2180,  2220,  2260,  2300,  2340,  2380,  2420,  2460,  2500,
         2540,  2580,  2620,  2660,  2700,  2740,  2780,  2820,  2860,
         2899,  2939,  2979,  3018,  3058,  3098,  3138,  3178,  3218,
         3258,  3298,  3338,  3378,  3418,  3458,  3498,  3538,  3578,
         3618,  3657,  3697,  3737,  3777,  3817,  3857,  3897,  3937,
         3977,  4017,  4056,  4096,  4136,  4176,  4215,  4255,  4295,
         4335,  4375,  4415,  4455,  4495,  4535,  4575,  4615,  4655,
         4695,  4735,  4775,  4815,  4854,  4894,  4934,  4974,  5014,
         5054,  5094,  5134,  5174,  5214,  5254,  5294,  5334,  5374,
         5414,  5454,  5494,  5534,  5574,  5614,  5654,  5694,  5734,
         5774,  5814,  5854,  5894,  5934,  5974,  6014,  6054,  6093,
         6133,  6173,  6213,  6253,  6293,  6333,  6373,  6413,  6453,
         6493,  6533,  6573,  6613,  6652,  6692,  6732,  6772,  6812,
         6852,  6892,  6931,  6970,  7010,  7050,  7090,  7130,  7170,
         7210,  7250,  7290,  7330,  7369,  7409,  7449,  7488,  7528,
         7568,  7608,  7648,  7688,  7728,  7768,  7808,  7848,  7888,
         7928,  7967,  8007,  8047,  8087,  8127,  8167,  8207,  8247,
         8287,  8325,  8365,  8405,  8445,  8485,  8511,  8551,  8591,
         8631,  8671,  8711,  8751,  8791,  8831,  8871,  8911,  8951,
         8991,  9031,  9071,  9111,  9151,  9191,  9231,  9271,  9311,
         9351,  9391,  9431,  9471,  9511,  9550,  9590,  9630,  9670,
         9710,  9750,  9790,  9830,  9870,  9910,  9950,  9990, 10030,
        10070, 10108, 10147, 10187, 10227, 10267, 10307, 10347, 10386,
        10425, 10465, 10505, 10545, 10585, 10625, 10663, 10703, 10743,
        10783, 10823, 10863, 10903, 10941, 10981, 11021, 11061, 11101,
        11141, 11181, 11221, 11261, 11301, 11341, 11381, 11421, 11461,
        11501, 11541, 11580, 11619, 11659, 11699, 11739, 11779, 11819,
        11858, 11898, 11938, 11978, 12018, 12058, 12098, 12138, 12177,
        12217, 12257, 12297, 12337, 12377, 12417, 12457, 12497, 12537,
        12577, 12617, 12657, 12697, 12737, 12777, 12817, 12856, 12896,
        12934, 12974, 13014, 13052, 13092, 13131, 13170, 13210, 13250,
        13289, 13328, 13368, 13408, 13448, 13488, 13528, 13568, 13608,
        13648, 13688, 13727, 13767, 13807, 13847, 13887, 13927, 13967,
        14007, 14046, 14086, 14125, 14165, 14205, 14245, 14285, 14325,
        14364, 14404, 14444, 14483, 14523, 14563, 14603, 14643, 14682,
        14722, 14761, 14801, 14841, 14881, 14921, 14960, 15000, 15040,
        15080, 15119, 15159, 15199, 15239, 15278, 15317, 15356, 15395,
        15435, 15475, 15515, 15554, 15593, 15633, 15673, 15713, 15753,
        15793, 15833, 15873, 15913, 15952, 15992, 16032, 16072, 16112,
        16152, 16191, 16231, 16270, 16310, 16350, 16390, 16430, 16470,
        16510, 16549, 16589, 16628, 16668, 16707, 16747, 16786, 16826,
```

```
        16866, 16906, 16946, 16985, 17025, 17065, 17105, 17145, 17185,
        17225, 17264, 17304, 17344, 17384, 17423, 17463, 17503, 17542,
        17580, 17620, 17660, 17700, 17740, 17780, 17820, 17860, 17899],
       dtype=int64),
 array([8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8], dtype=int64))
```

In [57]:
```python
#Removing outliers
df_new=df[(z<3).all(axis=1)]
df_new
```

Out[57]:

0	Bran5	Varia16	Moc30	2013	Transmission1	Fue2	KM_63000.0	Locatio7	Selli340000.0
1	0	36	1	2016	0	2	72000.0	17	2650000.0
2	8	18	2	2016	1	2	37739.0	1	295000.0
3	8	36	24	2007	1	2	38000.0	4	285000.0
4	8	3	25	2015	1	2	51000.0	21	470000.0
...	...	...	...	...	...	...	...	...	...
17995	4	35	6	2016	1	2	58058.0	14	675000.0
17996	8	30	5	2018	1	1	48000.0	18	799999.0
17997	5	0	18	2009	1	2	38000.0	20	160000.0
17998	8	26	29	2010	1	2	64000.0	24	160000.0
17999	11	8	19	2006	1	1	124000.0	26	330000.0

17463 rows × 9 columns

In [58]:
```python
#Original data dimensions
df.shape
```

Out[58]: (17913, 9)

In [59]:
```python
#New data dimensions
df_new.shape
```

Out[59]: (17463, 9)

Percentage loss of data after removing outliers :

In [60]:
```
dfshape = 17913
dfnewshape = 17463
total = dfshape-dfnewshape
percentage_loss=((total)/dfshape)*100
print(percentage_loss)
```

```
2.5121420197621838
```

Preparing dataset for model training :

In [61]:
```
df_x=df_new.drop('Selling_Price',axis=1) #Independent variables
y=df_new['Selling_Price'] #Target Variable
```

In [62]:
```
#Checking x data
df_x.head()
```

Out[62]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Location
0	5	16	30	2013	1	2	63000.0	7
1	0	36	1	2016	0	2	72000.0	17
2	8	18	2	2016	1	2	37739.0	1
3	8	36	24	2007	1	2	38000.0	4
4	8	3	25	2015	1	2	51000.0	21

In [63]:
```
#Checking y data after splitting
y.head()
```

Out[63]:
```
0     340000.0
1    2650000.0
2     295000.0
3     285000.0
4     470000.0
Name: Selling_Price, dtype: float64
```

Treating skewness :

In [64]:
```
#We are treating skewness by using square root transform
for col in df_x.skew().index:
    if col in df_x.describe().columns:
        if df_x[col].skew()>0.55:
            df_x[col]=np.sqrt(df_x[col])
        if df_x[col].skew()<-0.55:
            df_x[col]=np.sqrt(df_x[col])
```

In [65]:
```
#Checking skewness after treating it
df_x.skew()
```

Out[65]:
```
Brand          -0.251332
Variant         0.067599
Model          -0.067264
Year           -0.059469
Transmission   -1.916301
Fuel           -1.826748
```

```
KM_Driven       0.195224
Location       -0.081364
dtype: float64
```

Scaling the data :

In [66]:
```
#Scaling the dataset using StandardScaler
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(df_x)
x=pd.DataFrame(x,columns=df_x.columns)
x
```

Out[66]:

	Brand	Variant	Model	Year	Transmission	Fuel	KM_Driven	Location
0	-0.533280	-0.213255	1.631385	-0.058207	0.426813	0.728397	0.009725	-0.915092
1	-2.185166	1.573943	-1.667611	0.785805	-2.342949	0.728397	0.282987	0.223589
2	0.457851	-0.034535	-1.553852	0.785805	0.426813	0.728397	-0.884837	-1.598301
3	0.457851	1.573943	0.948834	-1.746232	0.426813	0.728397	-0.874263	-1.256697
4	0.457851	-1.374934	1.062593	0.504468	0.426813	0.728397	-0.387096	0.679062
...	...	...	...	...	...	...	...	...
17458	-0.863657	1.484583	-1.098818	0.785805	0.426813	0.728397	-0.148676	-0.118015
17459	0.457851	1.037784	-1.212577	1.348480	0.426813	-0.353225	-0.493416	0.337457
17460	-0.533280	-1.643014	0.266283	-1.183557	0.426813	0.728397	-0.874263	0.565194
17461	0.457851	0.680344	1.517627	-0.902220	0.426813	0.728397	0.041012	1.020666
17462	1.448983	-0.928134	0.380042	-2.027570	0.426813	-0.353225	1.604474	1.248403

17463 rows × 8 columns

Model Building :

In [67]:
```
#Importing required metrices and model for the dataset
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression,Lasso,ElasticNet,Ridge
from sklearn.metrics import r2_score, mean_absolute_error,mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoost
from sklearn.model_selection import GridSearchCV
```

In [68]:
```
#Finding the best random state and r2_score
for i in range(42,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=i)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pred_test)*1
        print('At random state',i,',the model performs well')
        print('Training r2_score is: ',r2_score(y_train,pred_train)*100)
        print('Testing r2_score is: ',r2_score(y_test,pred_test)*100)
```

```
At random state 54 ,the model performs well
```

```
Training r2_score is:  70.07096576957927
Testing r2_score is:  70.13238349195949
At random state 58 ,the model performs well
Training r2_score is:  70.08423009151878
Testing r2_score is:  70.10140926126549
At random state 60 ,the model performs well
Training r2_score is:  70.08257376435378
Testing r2_score is:  70.09190767021569
At random state 68 ,the model performs well
Training r2_score is:  70.09454430454532
Testing r2_score is:  70.0619973058046
At random state 83 ,the model performs well
Training r2_score is:  70.09747101108337
Testing r2_score is:  70.05752733796045
At random state 88 ,the model performs well
Training r2_score is:  70.08974187727294
Testing r2_score is:  70.05907571962359
```

In [69]:
```
#Creating train_test_split using best random_state
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=60,test_size=.20)
```

Finding the best model :

In [70]:
```
LR=LinearRegression()
l=Lasso()
en=ElasticNet()
rd=Ridge()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
rf=RandomForestRegressor()
ab=AdaBoostRegressor()
gb=GradientBoostingRegressor()
```

In [71]:
```
models= []
models.append(('Linear Regression',LR))
models.append(('Lasso Regression',l))
models.append(('Elastic Net Regression',en))
models.append(('Ridge Regression',rd))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
models.append(('RandomForestRegressor',rf))
models.append(('AdaBoostRegressor',ab))
models.append(('GradientBoostingRegressor',gb))
```

In [72]:
```
#Finding the required metrices for all models together using a for loop
Model=[]
score=[]
cvs=[]
sd=[]
mae=[]
mse=[]
rmse=[]
for name,model in models:
    print('******************************',name,'**************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=r2_score(y_test,pre)
    print('r2_score: ',AS)
    score.append(AS*100)
```

```
    print('\n')
    std=cross_val_score(model,x,y,cv=5,scoring='r2').std()
    print('Standard Deviation: ',std)
    sd.append(std)
    print('\n')
    MAE=mean_absolute_error(y_test,pre)
    print('Mean Absolute Error: ',MAE)
    mae.append(MAE)
    print('\n')
    MSE=mean_squared_error(y_test,pre)
    print('Mean Squared Error: ',MSE)
    mse.append(MSE)
    print('\n')
    RMSE=np.sqrt(mean_squared_error(y_test,pre))
    print('Root Mean Squared Error: ',RMSE)
    rmse.append(RMSE)
    print('\n\n')
```

```
****************************** Linear Regression ***************************

LinearRegression()

r2_score:  0.7009190767021569

Standard Deviation:  0.004176297129112747

Mean Absolute Error:  186690.21999500436

Mean Squared Error:  75072876505.13025

Root Mean Squared Error:  273994.300132558


****************************** Lasso Regression ***************************

Lasso()

r2_score:  0.7009187514017794

Standard Deviation:  0.004176488155667048

Mean Absolute Error:  186689.664186288

Mean Squared Error:  75072958159.40227

Root Mean Squared Error:  273994.4491397632


****************************** Elastic Net Regression ***************************

ElasticNet()
```

r2_score: 0.6035359531868805

Standard Deviation: 0.010418121774755689

Mean Absolute Error: 198821.12051002315

Mean Squared Error: 99517201220.77126

Root Mean Squared Error: 315463.470501374

****************************** Ridge Regression **************************

Ridge()

r2_score: 0.7009166171917083

Standard Deviation: 0.004178969373923639

Mean Absolute Error: 186683.6670608455

Mean Squared Error: 75073493871.56787

Root Mean Squared Error: 273995.42673476844

****************************** Decision Tree Regressor **************************

DecisionTreeRegressor()

r2_score: 0.9850460393947519

Standard Deviation: 0.004318084828754123

Mean Absolute Error: 26412.785451393756

Mean Squared Error: 3753622348.7660604

Root Mean Squared Error: 61266.81278445991

****************************** KNeighbors Regressor **************************

KNeighborsRegressor()

r2_score: 0.9839368925798914

```
Standard Deviation:  0.0057113044059591

Mean Absolute Error:  28907.33724592041

Mean Squared Error:  4032031419.2608876

Root Mean Squared Error:  63498.27886849287


****************************** RandomForestRegressor **************************

RandomForestRegressor()

r2_score:  0.9850450182455533

Standard Deviation:  0.004282132774484586

Mean Absolute Error:  26405.14046945697

Mean Squared Error:  3753878669.386033

Root Mean Squared Error:  61268.90458777628


****************************** AdaBoostRegressor **************************

AdaBoostRegressor()

r2_score:  0.9107886626550117

Standard Deviation:  0.006218416463179586

Mean Absolute Error:  113764.21943803274

Mean Squared Error:  22393108987.05561

Root Mean Squared Error:  149643.2724416825


****************************** GradientBoostingRegressor **************************

GradientBoostingRegressor()

r2_score:  0.9848465916269065

Standard Deviation:  0.0040595520556825195
```

```
Mean Absolute Error:  27887.261474512907

Mean Squared Error:  3803686115.721113

Root Mean Squared Error:  61674.031129164185
```

In [73]:
```
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'r2_score': score, 'Standard_deviation':sd,
                     'Mean_absolute_error':mae, 'Mean_squared_error':mse, 'Root_Mean_
result
```

Out[73]:

	Model	r2_score	Standard_deviation	Mean_absolute_error	Mean_squared_erro
0	Linear Regression	70.091908	0.004176	186690.219995	7.507288e+1
1	Lasso Regression	70.091875	0.004176	186689.664186	7.507296e+1
2	Elastic Net Regression	60.353595	0.010418	198821.120510	9.951720e+1
3	Ridge Regression	70.091662	0.004179	186683.667061	7.507349e+1
4	Decision Tree Regressor	98.504604	0.004318	26412.785451	3.753622e+0
5	KNeighbors Regressor	98.393689	0.005711	28907.337246	4.032031e+0
6	RandomForestRegressor	98.504502	0.004282	26405.140469	3.753879e+0
7	AdaBoostRegressor	91.078866	0.006218	113764.219438	2.239311e+1
8	GradientBoostingRegressor	98.484659	0.004060	27887.261475	3.803686e+0

Hyperparameter Tuning :

Random Forest Regressor

In [74]:
```
#Creating parameter list to pass in GridSearchCV
parameters={'criterion':['mse','mae'],'n_estimators':[50,100,500],'max_features':['a
```

In [75]:
```
#Using GridSearchCV to run the parameters and checking final accuracy
rf=RandomForestRegressor()
grid=GridSearchCV(rf,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_) #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'criterion': 'mse', 'max_features': 'auto', 'n_estimators': 50}
0.984574811251329
```

In [76]:
```
#Using the best parameters obtained
RF=RandomForestRegressor(random_state=48, n_estimators=500, criterion='mse', max_fea
RF.fit(x_train,y_train)
pred=RF.predict(x_test)
print('r2_score: ',r2_score(y_test,pred)*100)
print('Standard deviation: ',cross_val_score(RF,x,y,cv=5,scoring='r2').std())
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  98.50456691899329
Standard deviation:  0.004344321397752803
Mean absolute error:  26420.99733115128
Mean squared error:  3753715274.588142
Root Mean squared error:  61267.57114973746
```

Finalizing the model :

In [77]:
```
rf_prediction=RF.predict(x)
print('Predictions of Random Forest Regressor: ',rf_prediction)
```

```
Predictions of Random Forest Regressor:  [ 329216.98054833 2389843.39078276  317941.
42744627 ...  137383.0514037
  162438.04850012  280580.94305981]
```

In [78]:
```
#Comparing actual and predicted values with the help of a dataframe
predictions=pd.DataFrame({'Original_price':y, 'Predicted_price':rf_prediction})
predictions
```

Out[78]:

	Original_price	Predicted_price
0	340000.0	3.292170e+05
1	2650000.0	2.389843e+06
2	295000.0	3.179414e+05
3	285000.0	2.853192e+05
4	470000.0	4.672959e+05
...	...	...
17995	675000.0	6.651467e+05
17996	799999.0	7.799048e+05
17997	160000.0	1.373831e+05
17998	160000.0	1.624380e+05
17999	330000.0	2.805809e+05

17463 rows × 2 columns

Saving the model :

In [79]:
```
#Saving the model
import pickle
filename='Car_Price_Prediction.pkl'   #Specifying the filename
pickle.dump(RF,open(filename,'wb'))
```

In [80]:
```
#Saving the predicted values
results=pd.DataFrame(rf_prediction)
results.to_csv('Car_Price_Prediction_Results.csv')
```

Conclusion:

Key Findings and Conclusions of the Study:

We collected the used cars data from olx and it was done by using Web scraping. The framework used for web scraping was Selenium, which has an advantage of automating our process of collecting data. We collected almost 18000 of data which contained the selling price and other related features.Then, the scrapped data was combined in a single data frame and saved in a csv file so that we can open it and analyse the data.We did data cleaning, data-preprocessing steps like finding and handling null values, removing words from numbers, converting object to int type, data visualization, handling outliers, etc.After separating our train and test data, we started running different machine learning classification algorithms to find out the best performing model.We found that RandomForest and KNeighbors Algorithms were performing well, according to their r2_score and cross val scores.Then, we performed Hyperparameter Tuning techniques using GridSearchCV for getting the best parameters and improving the scores. In that, RandomForestRegressor performed well and we finalised that model.We saved the model in pkl format and then saved the predicted values in a csv format.

The problems we faced during this project were:

No information for handling these fast-paced websites were informed so that we were consuming more time in web scraping itself. Many outliers were removed as some of the selling price and km values were out of range.

In [ ]:

In [ ]: