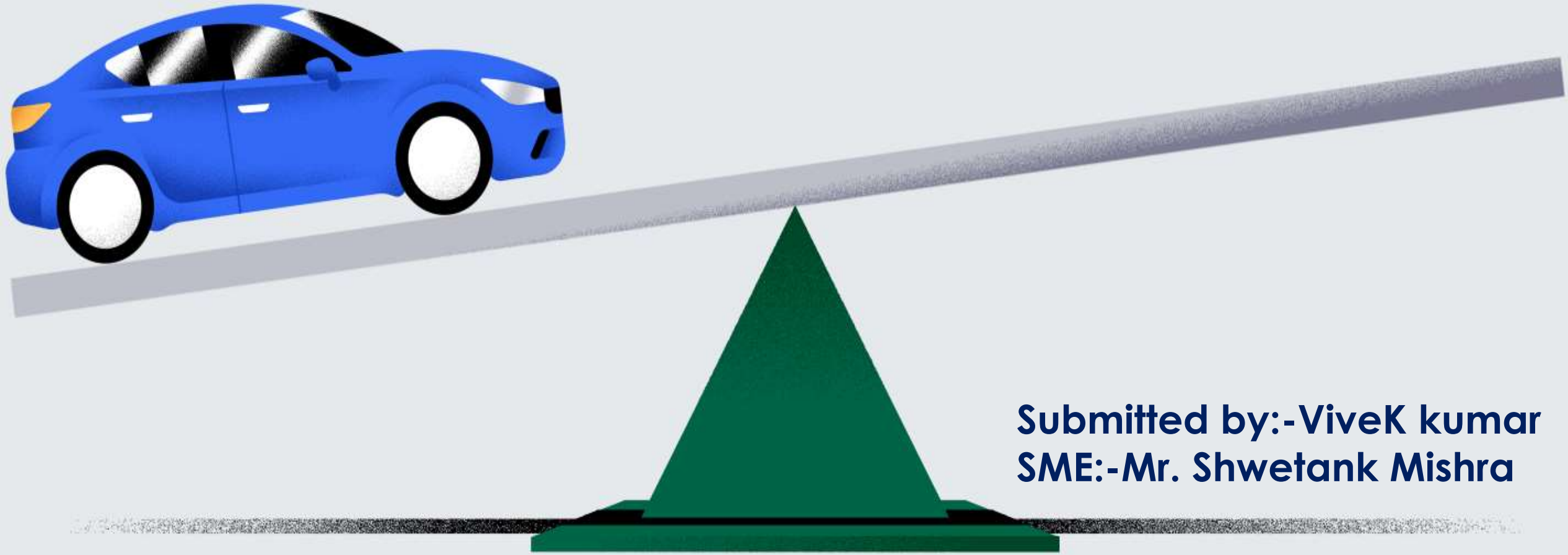
CAR PRICE PREDICTION



Submitted by:-ViveK kumar
SME:-Mr. Shwetank Mishra

AGENDA

- Introduction
- Problem Statement
- Business Goal
- Technical Requirement
- Exploratory Data Analysis (EDA)
- Data Pre-Processing
- Visualization
- Built Model
- Saved Best Model
- Summary

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.

This project contains two phase:

- Data Collection Phase : We have to scrape at least 5000 used cars data. We can scrape more data as well, it's up to us. More the data better the model. In this section we need to scrape the data of used cars from websites (Olx, cardekho, Cars24 etc.) We need web scraping for this. We have to fetch data for different locations. The number of columns for data doesn't have limit, it's up to us and our creativity. Generally, these columns are Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location and at last target variable Price of the car. This data is to give us a hint about important variables in used car model. We can make changes to it, we can add or you can remove some columns, it completely depends on the website from which we are fetching the data. Try to include all types of cars in our data for example- SUV, Sedans, Coupe, minivan, Hatchback.
- Model Building Phase : After collecting the data, you 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. Follow the complete life cycle of data science

BUSINESS GOAL

- 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. This project contains two phases:
  - Data Collection Phase
  - Model Building Phase

TECHNICAL REQUIREMENT

We have to made car price valuation model. This project contains two phases:

- **Data Collection Phase**: We have scraped more than 5000 used cars data from websites: cardekho, Olx and cars24. We have fetched data for different locations. All types of cars are present in data for example- SUV, Sedans, Coupe, minivan, Hatchback. After scraping converted into csv file.
- **Model Building Phase**: After collecting the data, built a machine learning model. Before model building have done all data pre-processing steps. Tried different models with different hyper parameters and selected the best model. Followed the complete life cycle of data science. Include all the steps like:

EXPLORATORY DATA ANALYSIS (EDA)

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Checked for special character present in dataset or not
- Checked total number of unique value
- Dropped irrelevant Features
- Replaced duplicate values, special characters and irrelevant data
- Checked Information about Data
- Checked all features through visualization.

DATA DESCRIPTION

The dataset contains 5616 records (rows) and 10 features (columns).

```
car.columns
```

```
Index(['Unnamed: 0', 'Brand', 'Model', 'Variant', 'Manufacturing_Year',
       'Driven_KiloMeters', 'Fuel', 'Location', 'Car_Price',
       'Number_of_Owners'],
      dtype='object')
```

After removal of irrelevant data and column, we remains with 5483 records (rows) and 8 features (columns).

```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5483 entries, 0 to 5482
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Brand                 5483 non-null   object 
 1   Model                 5483 non-null   object 
 2   Variant               5483 non-null   object 
 3   Manufacturing_Year    5483 non-null   int64  
 4   Driven_KiloMeters     5483 non-null   object 
 5   Fuel                  5483 non-null   object 
 6   Location              5483 non-null   object 
 7   Car_Price             5483 non-null   float64
```

DATASET DESCRIPTION

```
car.describe()
```

	Manufacturing_Year	Car_Price
count	5483.000000	5.483000e+03
mean	2016.043407	8.558486e+05
std	3.333458	9.607719e+05
min	2000.000000	4.000000e+04
25%	2015.000000	4.150000e+05
50%	2017.000000	6.017990e+05
75%	2018.000000	9.250000e+05
75%	2018.000000	9.250000e+05
max	2022.000000	9.900000e+06

Checking Description through heatmap also

```
plt.figure(figsize=(10,5))
sns.heatmap(round(car.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt='.2f')
plt.xticks(fontsize=18)
plt.xticks(fontsize=12)
plt.title('variables')
plt.show()
```

Observation of Describe of Datasets:

- The summary of this dataset looks good as there are no negative/ invalid value present.
- We can see the counts of "Manufacturing_Year" and "Car_Price" columns are 5483.000000 which means no null values are present.
- Total No of Rows: 5483 and Total No. of Columns: 8
- Only two column contains Continuous Data that is "Manufacturing_Year".
- We are determining Mean, Standard Deviation, Minimum and Maximum Values of each column.

DATA CLEANING

Handling Null Values

```
#Droping column "Number_of_Owners" as it contains most missing value(3863) which is more than 50%.
car.drop(columns=['Number_of_Owners'],inplace=True)
```

```
#Droping column "Unnamed: 0"
car.drop(columns=['Unnamed: 0'],inplace=True)
```

```
#We cannot fill any value in Variant as it is unique for each model, and also we cann't drop column as it have very less missing
car.dropna(inplace = True)
```

```
car.reset_index(inplace=True, drop=True)
```

```
#Brand column:
car["Brand"]= car["Brand"].str.replace('Bmw', 'BMW')
car["Brand"]= car["Brand"].str.replace('Kia', 'KIA')
```

```
#Driven_KiloMeters column:
car["Driven_KiloMeters"]= car["Driven_KiloMeters"].str.replace('KM', ' ')
```

```
car["Driven_KiloMeters"]= car["Driven_KiloMeters"].str.replace('kms', '')
```

```
car["Driven_KiloMeters"]= car["Driven_KiloMeters"].str.replace('km', '')
```

```
car["Driven_KiloMeters"]= car["Driven_KiloMeters"].str.replace(',', '')
```

```
car["Driven_KiloMeters"]= car["Driven_KiloMeters"].str.replace('.0', '')
```

```
#Fuel column:
car["Fuel"].replace('Petrol', 'PETROL',inplace=True)
```

```
#Fuel column:
car["Fuel"].replace('Diesel', 'DIESEL',inplace=True)
```

```
#Location column:
car["Location"].replace('Delhi NCR', 'Delhi',inplace=True)
```

```
#Car_Price column:
car["Car_Price"]= car["Car_Price"].str.replace('₹', '')
car["Car_Price"]= car["Car_Price"].str.replace('.', '')
car["Car_Price"]= car["Car_Price"].str.replace(',', '')
car["Car_Price"]= car["Car_Price"].str.replace(' Lakh', '000')
```

DATA VISUALIZATION

1. Univariate Analysis
   - ✓ Using Countplot

2. Bivariate Analysis (for comparison between features and target)
   - ✓ Using Catplot and Scatterplot

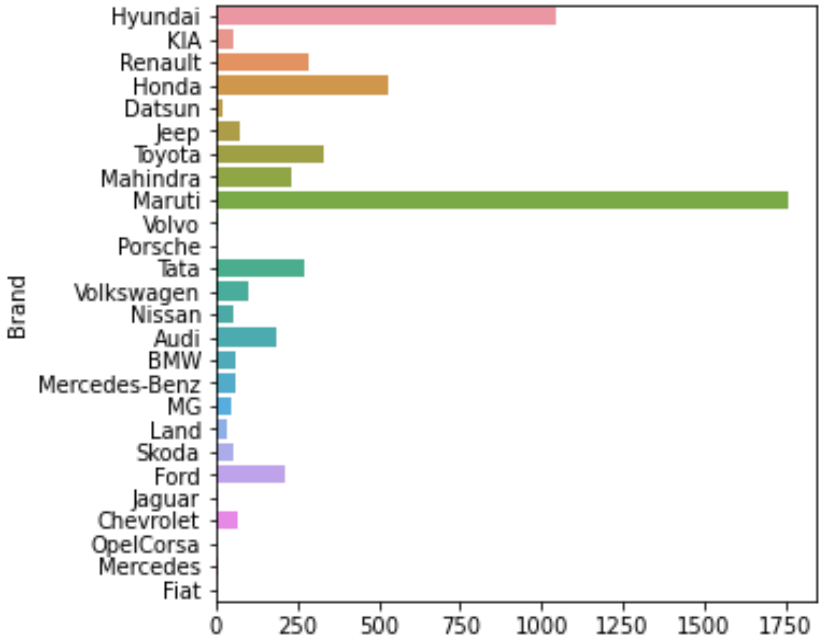
3. Multivariate Analysis
   - ✓ Using Pairplot (comparison between all continuous features and target)

Univariate Analysis

Using Countplot

```
#Count Plot for "Brand" column
plt.figure(figsize=(5,5))
sns.countplot(y="Brand",data=car)
```

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

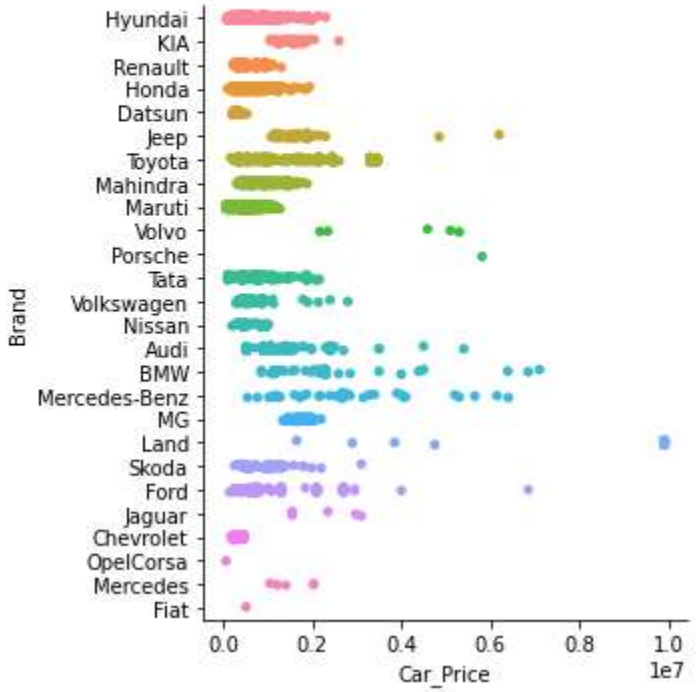


Bivariate Analysis

Using Catplot and Scatterplot

```
plt.figure(figsize=(25,20))
sns.catplot(x='Car_Price', y= 'Brand', data = car)
plt.show()
```

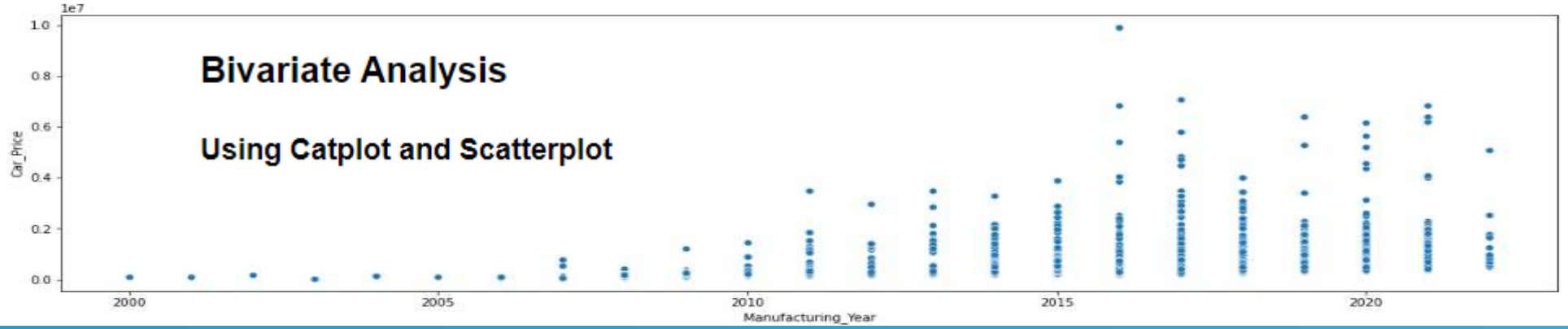
```
<Figure size 1800x1440 with 0 Axes>
```



Here we can see that Maruti Brand car is available most (Total No= 1756) compare to other Brand

```
#scatterplot for comparision between "Manufacturing_Year" and "Car_Price" column
plt.figure(figsize=(20,5))
sns.scatterplot(x="Manufacturing_Year",data=car, y='Car_Price')
```

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

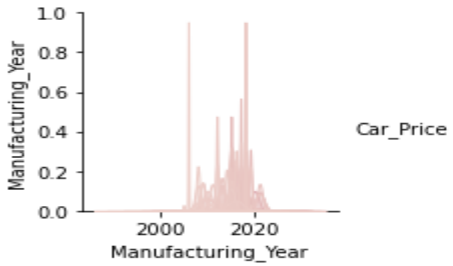


Multivariate Analysis

Using pairplot

```
sns.pairplot(car,hue="Car_Price")
```

```
<seaborn.axisgrid.PairGrid at 0x1e12c0de910>
```



We can observe relationship between all the continuous column and the target column by this pairplot in pairs which are plotted on basis of target column.

CHECKING CORRELATION

```
car.corr()
```

	Brand	Model	Variant	Manufacturing_Year	Driven_KiloMeters	Fuel	Location	Car_Price
Brand	1.000000	0.984207	0.123862	0.129157	-0.028763	0.015162	0.000459	-0.003022
Model	0.984207	1.000000	0.106493	0.088589	-0.023588	-0.013179	0.044661	0.002470
Variant	0.123862	0.106493	1.000000	0.005068	-0.072386	0.169128	-0.073757	-0.086074
Manufacturing_Year	0.129157	0.088589	0.005068	1.000000	-0.160688	-0.005101	-0.151555	0.231122
Driven_KiloMeters	-0.028763	-0.023588	-0.072386	-0.160688	1.000000	-0.166733	-0.019620	-0.085451
Fuel	0.015162	-0.013179	0.169128	-0.005101	-0.166733	1.000000	-0.122324	-0.309870
Location	0.000459	0.044661	-0.073757	-0.151555	-0.019620	-0.122324	1.000000	0.032826
Car_Price	-0.003022	0.002470	-0.086074	0.231122	-0.085451	-0.309870	0.032826	1.000000

This gives the correlation between the denpendent and independent variables.

```
car.corr()["Car_Price"].sort_values()
```

```
Fuel                -0.309870
Variant             -0.086074
Driven_KiloMeters   -0.085451
Brand               -0.003022
Model                0.002470
Location             0.032826
Manufacturing_Year   0.231122
Car_Price            1.000000
Name: Car_Price, dtype: float64
```

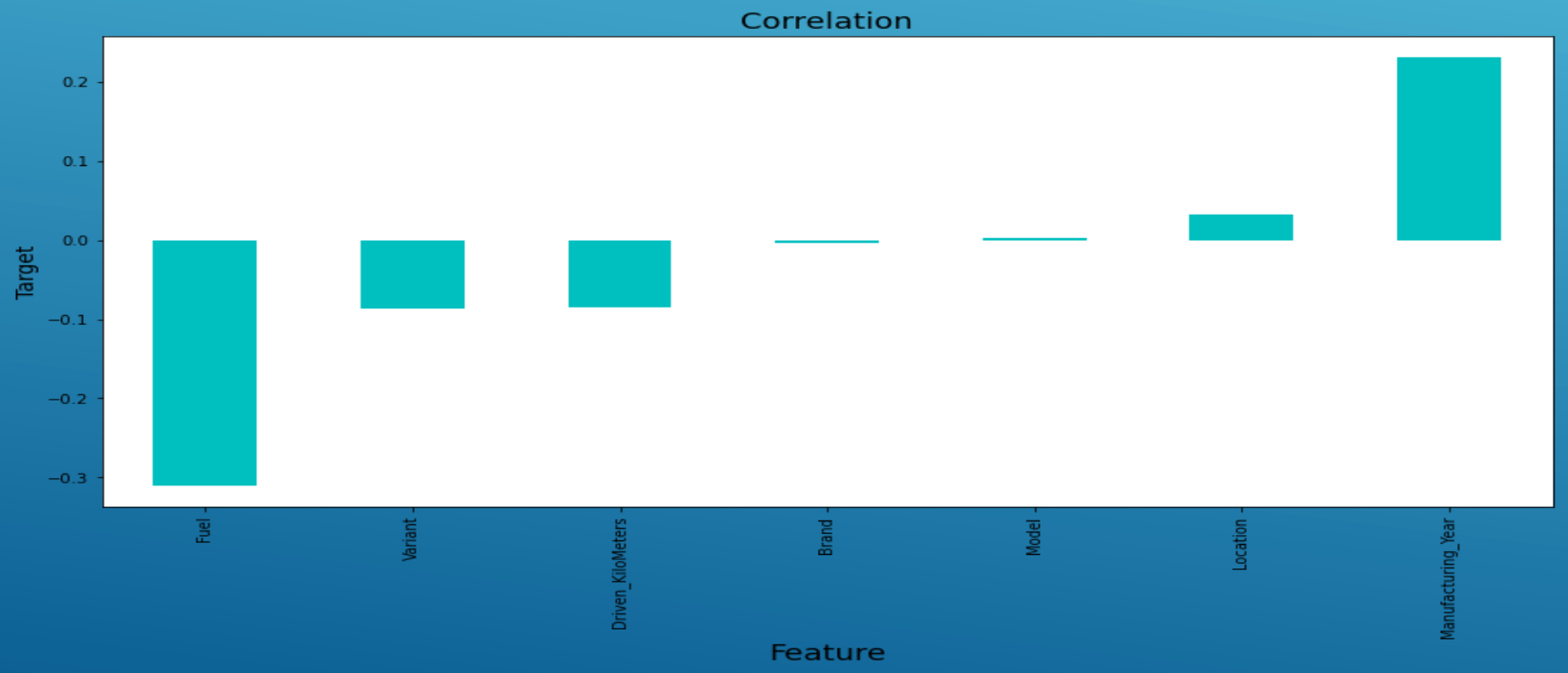
We can observe :

- All columns are sorted in ascending order showing least to strong correlation with target column.
- 3 columns are negatively correlated and 4 columns are positively correlated.
- Column 'Fuel' is highly positively correlated with Target column and Column 'Brand' is highly negatively correlated with Target column

- Correlation is checked for relation between the dependent and independent variables.
- Also Checked through heatmap and BarPlot (Visualization)
- Checking Correlation through Barplot :

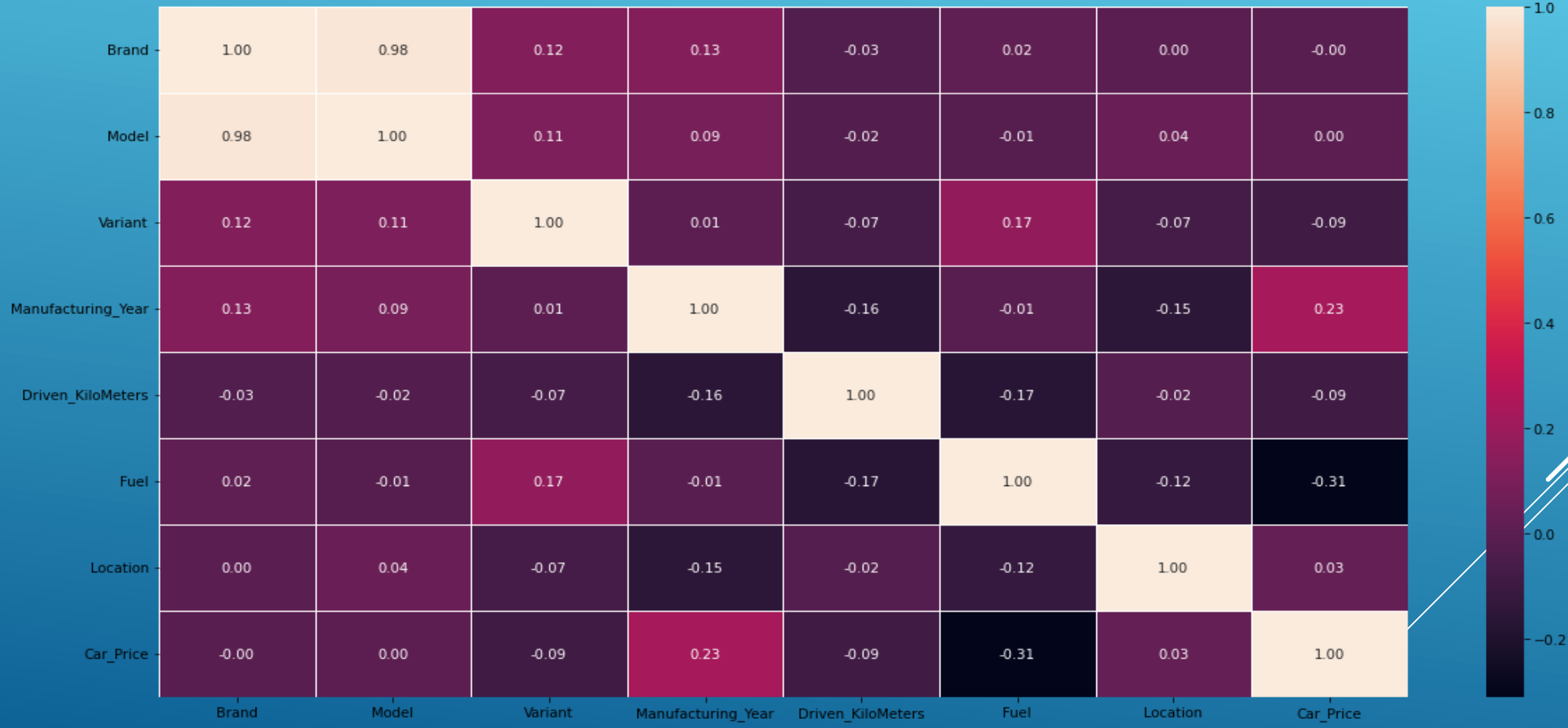
Checking correlation with barplot

```
plt.figure(figsize=(15,7))
car.corr()['Car_Price'].sort_values(ascending=True).drop(['Car_Price']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```



Checking correlation with heatmap

```
plt.figure(figsize=(20,10))
sns.heatmap(car.corr(),annot=True,annot_kws= {"size": 10}, linewidth=0.5, linecolor='white', fmt='.2f')
```



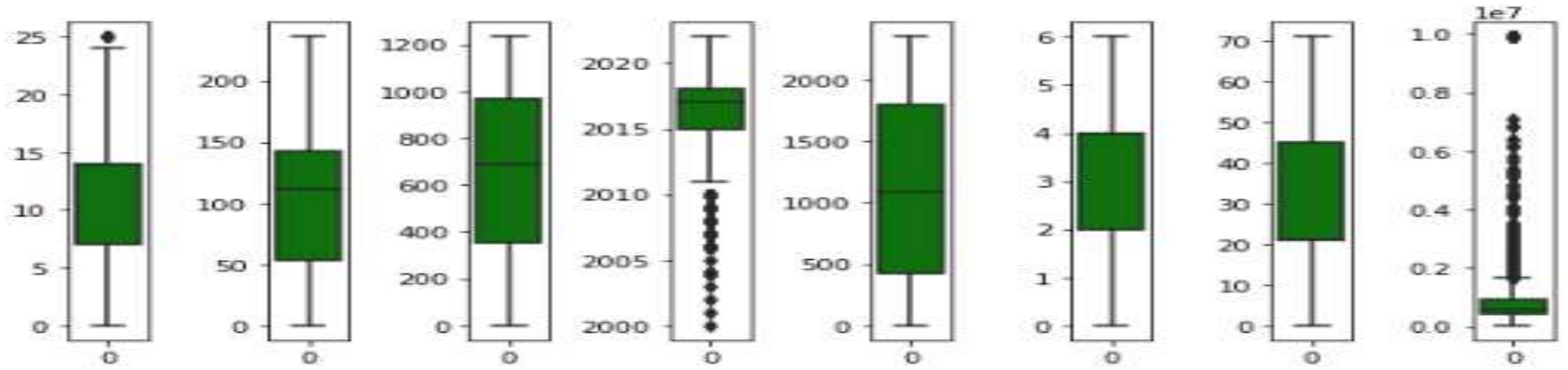
OUTCOME

Outcome of Correlation

- **Brand** has **-0 percent** correlation with the target column which can be considered as No correlation and is negatively correlated.
- **Model** has **-6 percent** correlation with the target column which can be considered as good correlation and negatively correlated.
- **Variant** has **10 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Manufacturing_Year** has **29 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Driven_KiloMeters** has **4 percent** correlation with the target column which can be considered as weak correlation and positively correlated.
- **Fuel** has **33 percent** correlation with the target column which can be considered as strong correlation and positively correlated.
- **Location** has **-21 percent** correlation with the target column which can be considered as weak correlation and negatively correlated.
  - Max correlation is with **Fuel**
  - Min correlation is with **Brand**

CHECKING OUTLIERS

```
collist=car.columns.values
ncol=8
nrows=7
plt.figure(figsize=(ncol,3*ncol))
for i in range(0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(data=car[collist[i]],color='green',orient='v')
    plt.tight_layout()
```



Observation:

- **Outliers present in columns:** "Brand", "Manufacturing_Year" and "Car_Price".
- But we will not remove Outliers from "Brand" column as it is categorical column and from "Car_Price" column as it is a target column.
- **Outliers not present in columns:** 'Model', 'Variant','Driven_KiloMeters', 'Fuel' and 'Location'.

REMOVING OUTLIERS

Outliers are removed only from continuous features and not from target

- Checking two methods and compare between them which is give less percentage loss and then using that method for further process.

1.Zscore method using Scipy

2.IQR (Inter Quantile Range) method

1.1 Zscore method using Scipy

```
#  Outliers will be removed only from Continuous column i.e; "Manufacturing_Year".
# We will not remove outliers from Categorical column i.e; "Brand".

variable = car[['Manufacturing_Year']]

z=np.abs(zscore(variable))

# Creating new dataframe
car_price = car[(z<3).all(axis=1)]
```

- **Comparing shape of old and new DataFrame after outliers removal**

```
print("Old DataFrame data in Rows and Column:",car.shape)
print("New DataFrame data in Rows and Column:",car_price.shape)
print("Total Dropped rows:",car.shape[0]-car_price.shape[0])
```

```
Old DataFrame data in Rows and Column: (5483, 8)
New DataFrame data in Rows and Column: (5413, 8)
Total Dropped rows: 70
```

2.1 IQR (Inter Quantile Range) method

```
#1st quantile
Q1=variable.quantile(0.25)

# 3rd quantile
Q3=variable.quantile(0.75)

#IQR
IQR=Q3 - Q1
car_price_pred=car[~((car < (Q1 - 1.5 * IQR)) |(car > (Q3 + 1.5 * IQR))).any(axis=1)]
```

- **Comparing shape of old and new DataFrame after outliers removal**

```
print("Old DataFrame data in Rows and Column:",car.shape)
print("\nNew DataFrame data in Rows and Column:",car_price_pred.shape)
print("\nTotal Dropped rows:",car.shape[0]-car_price_pred.shape[0])
```

```
Old DataFrame data in Rows and Column: (5483, 8)

New DataFrame data in Rows and Column: (5027, 8)

Total Dropped rows: 456
```

COMPARING DATA LOSS USING BOTH METHOD AFTER OUTLIER REMOVAL

1.2 Percentage Data Loss using Zscore

```
loss_percent=(5489-5419)/5489*100
print("loss_percent= ",loss_percent,"%")
```

```
loss_percent=  1.275277828384041 %
```

2.2 Percentage Data Loss using IQR

```
loss_perc = (5489-5033)/5489*100
print("loss_percent= ",loss_perc,"%")
```

```
loss_percent=  8.307524139187466 %
```

We can check by using IQR method there is large data loss in comparision to Zscore method. So, we will consider Zscore method.

CHECKING SKEWNESS

```
car_price.skew()
```

```
Brand                0.181949
Model                0.283178
Variant             -0.052374
Manufacturing_Year  -0.703623
Driven_KiloMeters   -0.068305
Fuel                -0.441923
Location             0.640418
Car_Price            5.656679
```

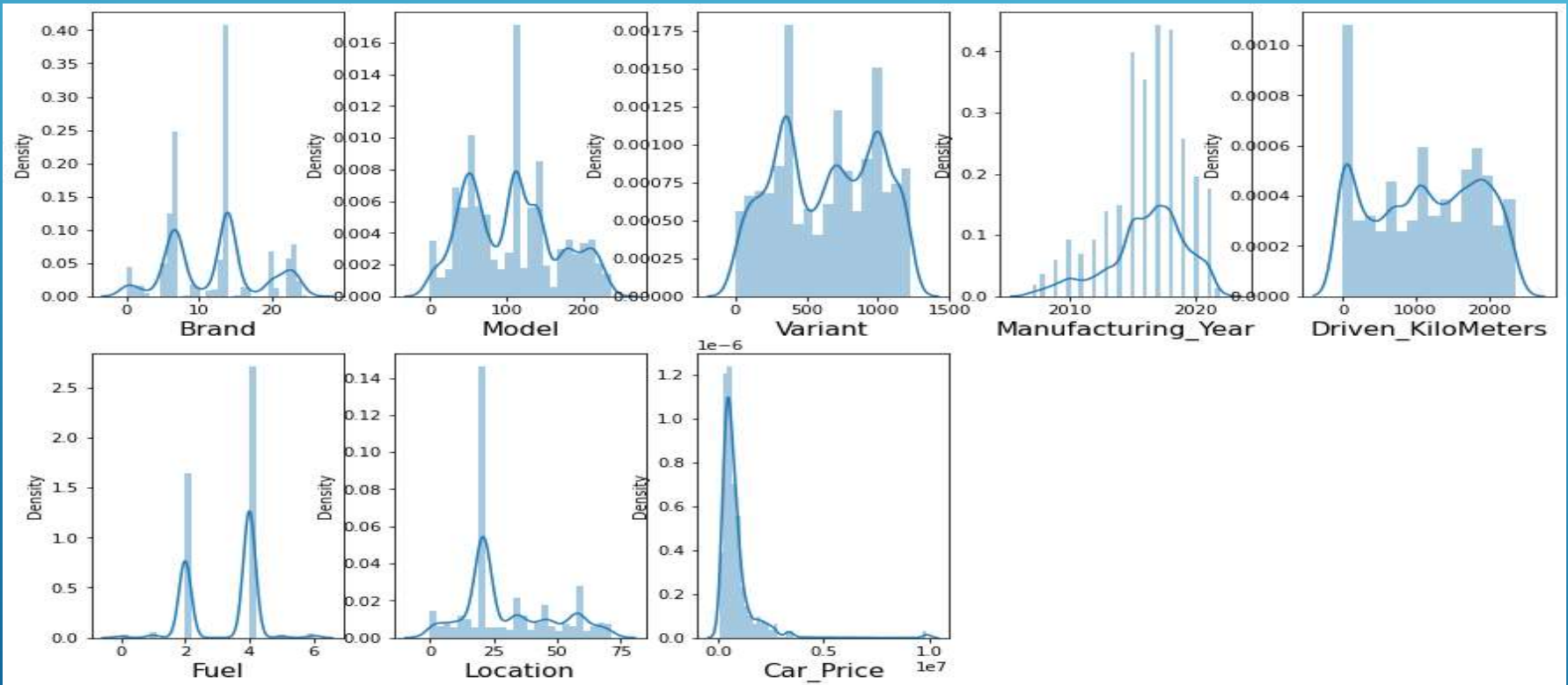
Observation:

- Skewness threshold taken is +/-0.25
- All the columns are not normallly distributed, they are skewed.
- Columns which are having skewness: 'Brand', 'Model', 'Manufacturing_Year', 'Fuel', 'Car_Price'.
- The 'Fuel' column data is negatively highly skewed and 'Location' is positively highly skewed
- Since 'Brand', 'Model', 'Fuel' are categorical column so we will not remove skewness and 'Car_Price' is Target Column so we can not remove skewness.
- So we will remove skewness from Manufacturing_Year column as it contains continuous data.

Checking skweness through Data Visualization also

```
plt.figure(figsize=(15,15), facecolor='white')
plotnumber = 1

for column in car_price:
    if plotnumber<=15:
        ax = plt.subplot(3,5,plotnumber)
        sns.distplot(car_price[column])
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.show()
```



REMOVING SKEWNESS

- Using yeo-Johnson method

```
collist=['Manufacturing_Year']
car_price[collist]=power_transform(car_price[collist],method='yeo-johnson')
car_price[collist]
```

	Manufacturing_Year
0	1.642232
1	0.922537
2	0.574776
3	-0.097454
4	1.278296
...	...
5478	-0.739757
5479	-0.097454
5480	-1.050095
5481	0.234837
5482	-0.097454

5413 rows × 1 columns

CHECKING SKEWNESS AFTER REMOVAL

```
car_price.skew()
```

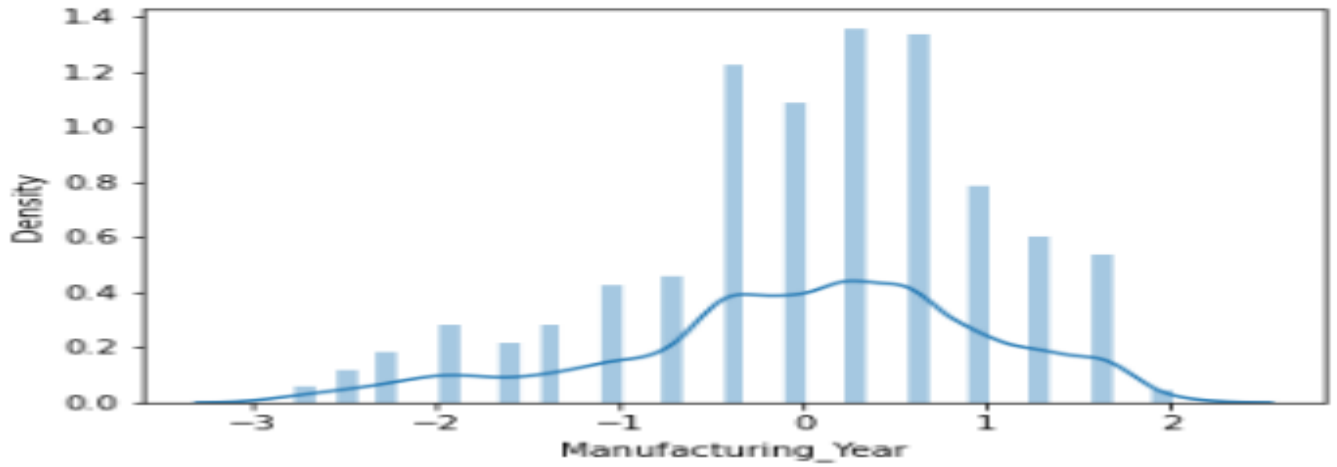
```
Brand                 0.181949
Model                 0.283178
Variant              -0.052374
Manufacturing_Year   -0.521735
Driven_KiloMeters    -0.068305
Fuel                 -0.441923
Location              0.640418
Car_Price             5.656679
dtype: float64
```

Still we can see skewness is present but from earlier it is removed.

Checking through Visualization

```
sns.distplot(car_price['Manufacturing_Year'])
```

```
<AxesSubplot:xlabel='Manufacturing_Year', ylabel='Density'>
```



DATA PRE-PROCESSING

Spliting data into Target and Features

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

```
x.head()
```

	Brand	Model	Variant	Manufacturing_Year	Driven_KiloMeters	Fuel	Location
0	7	71	625	1.642232	75	4	2
1	10	90	639	0.922537	332	4	2
2	20	176	751	0.574776	804	4	2
3	6	46	1211	-0.097454	549	4	2
4	3	28	917	1.278296	483	4	2

```
y.head()
```

```
0    2300000.0
1    1553000.0
2     711000.0
3     733000.0
4     381000.0
Name: Car_Price, dtype: float64
```

```
x.shape, y.shape
((5413, 7), (5413,))
```

Scaling data using Standard Scaler

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

```
x.head()
```

	Brand	Model	Variant	Manufacturing_Year	Driven_KiloMeters	Fuel	Location
0	-0.810184	-0.610846	-0.039789	1.642232	-1.413423	0.723823	-1.601946
1	-0.328258	-0.288392	0.000313	0.922537	-1.062234	0.723823	-1.601946
2	1.278161	1.171135	0.321125	0.574776	-0.417250	0.723823	-1.601946
3	-0.970826	-1.035127	1.638746	-0.097454	-0.765705	0.723823	-1.601946
4	-1.452752	-1.340609	0.796614	1.278296	-0.855894	0.723823	-1.601946

Checking for Multicolinearity

VIF (Variance Inflation factor)

```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	36.587438	Brand
1	36.172661	Model
2	1.047921	Variant
3	1.140527	Manufacturing_Year
4	1.075751	Driven_KiloMeters
5	1.090563	Fuel
6	1.096371	Location

The VIF value is more than 10 in the columns 'Brand', 'Model'. But column 'Brand' is having highest VIF value. So, we will drop column 'Brand'.

```
x = x.drop(['Brand'],axis=1)
```

```
#Checking again Multicolinearity using VIF
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.035566	Model
1	1.043016	Variant
2	1.090540	Manufacturing_Year
3	1.073187	Driven_KiloMeters
4	1.071377	Fuel
5	1.052502	Location

Now, we can check Multicolinearity is removed from the columns as VIF value of all columns are less than 10.

Variance Threshold Method

It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features.

```
var_threshold = VarianceThreshold(threshold=0)
var_threshold.fit(x)
```

```
VarianceThreshold(threshold=0)
```

```
var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True])
```

```
x.columns[var_threshold.get_support()]
```

```
Index(['Model', 'Variant', 'Manufacturing_Year', 'Driven_KiloMeters', 'Fuel',
       'Location'],
      dtype='object')
```

```
# taking  out  all the  constant  columns
cons_columns = [column for column in x.columns
                if column not in x.columns[var_threshold.get_support()]]
print(len(cons_columns))
```

```
0
```

So we can see that, with the help of variance threshold method, we got to know all the features here are important. So, we will create model now.

Creating Model

Finding the best random state among all the models

On the basis of target column as it contains continuous data, we will understand this by Regression Problem

```
maxAcc = 0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = i)
    modDTR =  DecisionTreeRegressor()
    modDTR.fit(x_train,y_train)
    pred = modDTR.predict(x_test)
    acc  = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print(f"Best Accuracy is: {maxAcc} on random_state: {maxRS}")
```

```
Best Accuracy is: 0.920262170460058 on random_state: 19
```

Creating train-test-split

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .25, random_state = maxRS)
```

Regression Algorithm

1. Linear Regression

```
# Checking r2score for Linear Regression
LR = LinearRegression()
LR.fit(x_train,y_train)

# prediction
predLR=LR.predict(x_test)
print('R2_score:',r2_score(y_test,predLR))
print('Mean abs error:',mean_absolute_error(y_test, predLR))
print('Mean squared error:',mean_squared_error(y_test, predLR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predLR)))
```

```
R2_score: 0.16172401078522514
Mean abs error: 425238.7765592483
Mean squared error: 797760765690.3414
Root Mean Squared Error:  893174.5437988823
```

2. Random forest Regression Model

```
# Checking R2 score for Random Forest Regressor
RFR=RandomForestRegressor(n_estimators=600, random_state=maxRS)
RFR.fit(x_train,y_train)

# prediction
predRFR=RFR.predict(x_test)
print('R2_Score:',r2_score(y_test,predRFR))
print('Mean abs error:',mean_absolute_error(y_test, predRFR))
print('Mean squared error:',mean_squared_error(y_test, predRFR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predRFR)))
```

```
R2_Score: 0.9514161523616257
Mean abs error: 84944.57991002557
Mean squared error: 46235712332.01829
Root Mean Squared Error:  215024.91095688957
```

3. KNN Regressor

```
# Checking R2 score for KNN regressor
knn=KNeighborsRegressor(n_neighbors=9 )
knn.fit(x_train,y_train)

#prediction
predknn=knn.predict(x_test)
print('R2_Score:',r2_score(y_test,predknn))
print('Mean abs error:',mean_absolute_error(y_test, predknn))
print('Mean squared error:',mean_squared_error(y_test, predknn))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predknn)))
```

```
R2_Score: 0.6336624049265533
Mean abs error: 251330.27835220745
Mean squared error: 348631911335.91406
Root Mean Squared Error:  590450.6002502784
```

4. Gradient boosting Regressor

```
# Checking R2 score for GBR
Gb= GradientBoostingRegressor(n_estimators=400,  random_state=maxRS, learning_rate=0.1, max_depth=3)
Gb.fit(x_train,y_train)

#prediction
predGb=Gb.predict(x_test)
print('R2_Score:',r2_score(y_test,predGb))
print('Mean abs error:',mean_absolute_error(y_test, predGb))
print('Mean squared error:',mean_squared_error(y_test, predGb))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predGb)))
```

```
R2_Score: 0.9550817155983341
Mean abs error: 102938.44305000136
Mean squared error: 42747311647.725655
Root Mean Squared Error:  206754.23006005381
```

5. Decision Tree Regressor

```
# Checking R2 score for GBR
DTR= DecisionTreeRegressor()
DTR.fit(x_train,y_train)

#prediction
predDTR=DTR.predict(x_test)
print('R2_Score:',r2_score(y_test,predDTR))
print('Mean abs error:',mean_absolute_error(y_test, predDTR))
print('Mean squared error:',mean_squared_error(y_test, predDTR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predDTR)))
```

```
R2_Score: 0.877309466408728
Mean abs error: 109852.28951255539
Mean squared error: 116760703252.8907
Root Mean Squared Error:  341702.65327165765
```

Cross Validation Score for all the model

```
#CV Score for Linear Regression
print('CV score for Linear Regression: ',cross_val_score(LR,x,y,cv=5).mean())

#CV Score for Random Forest Regression
print('CV score for Random forest Regression: ',cross_val_score(RFR,x,y,cv=5).mean())

#CV Score for KNN Regression
print('CV score for KNN Regression: ',cross_val_score(knn,x,y,cv=5).mean())

#CV Score for Gradient Boosting Regression
print('CV score for Gradient Boosting Regression: ',cross_val_score(Gb,x,y,cv=5).mean())

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

```
CV score for Linear Regression:  0.15707215392115464
CV score for Random forest Regression:  0.7464943437285952
CV score for KNN Regression:  0.32119277110194466
CV score for Gradient Boosting Regression:  0.7509483931060308
CV score for Decision Tree Regression:  0.5074509918030621
```

Hyper Parameter Tuning

The Gradient boosting regressor with GridsearchCV

```
parameter = {'n_estimators':[100,200,300,400],
             'learning_rate':[0.1,0.01,0.001,1],
             'subsample': [0.1,0.2,0.3,0.5,1],
             'max_depth':[1,2,3,4],
             'alpha':[0.1,0.01,0.001,1]}
```

```
CV_GBR = GridSearchCV(GradientBoostingRegressor(),parameter,cv=6,n_jobs = 3,verbose = 2)
```

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

```
Fitting 6 folds for each of 1280 candidates, totalling 7680 fits

GridSearchCV(cv=6, estimator=GradientBoostingRegressor(), n_jobs=3,
             param_grid={'alpha': [0.1, 0.01, 0.001, 1],
                         'learning_rate': [0.1, 0.01, 0.001, 1],
                         'max_depth': [1, 2, 3, 4],
                         'n_estimators': [100, 200, 300, 400],
                         'subsample': [0.1, 0.2, 0.3, 0.5, 1]},
             verbose=2)
```

```
CV_GBR.best_params_
```

```
{'alpha': 0.001,
 'learning_rate': 0.1,
 'max_depth': 4,
 'n_estimators': 400,
 'subsample': 0.5}
```

Creating Regressor Model with Gradient Boosting Regressor

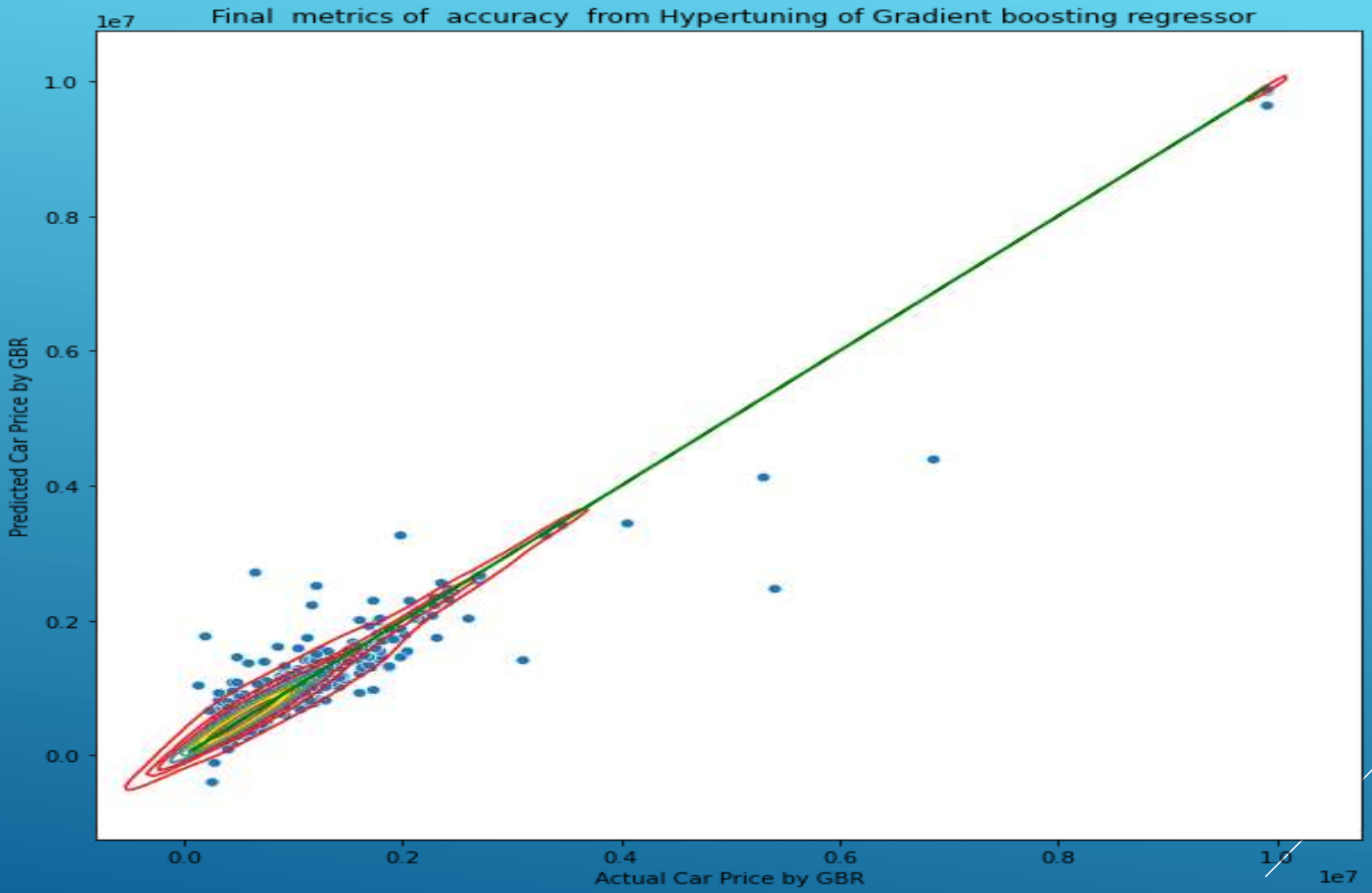
```
GBR = GradientBoostingRegressor(n_estimators=400, alpha=0.001,learning_rate= 0.1, max_depth= 4, subsample = 0.5)
GBR.fit(x_train, y_train)
```

```
GradientBoostingRegressor(alpha=0.001, max_depth=4, n_estimators=400,
                          subsample=0.5)
```

```
#prediction
GBRpred = GBR.predict(x_test)
#R2 score
acc = r2_score(y_test,GBRpred)
print(acc*100)
```

```
95.60799200909335
```

```
#Verifying the final  performance of the model by graph
plt.figure(figsize=(10,10))
sns.scatterplot(x=y_test,y=GBRpred,palette='Set2')
sns.kdeplot(x=y_test,y=GBRpred, cmap='Set1')
plt.plot(y_test,y_test,color='g')
#Verifying the performance of the model by graph
plt.xlabel("Actual Car Price by GBR")
plt.ylabel("Predicted Car Price by GBR")
plt.title(" Final  metrics of  accuracy  from Hypertuning of Gradient boosting regressor")
plt.show()
```


Final metrics of accuracy from Hypertuning of Gradient boosting regressor
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Predicted Car Price by GBR
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Actual Car Price by GBR

FINAL PROCEDURE:

1. SAVING THE MODEL

```
#saving the model at local file system
filename='Car_price_prediction.pickle'
pickle.dump(CV_GBR,open(filename,'wb'))
#prediction using the saved model
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.predict(x_test)
```

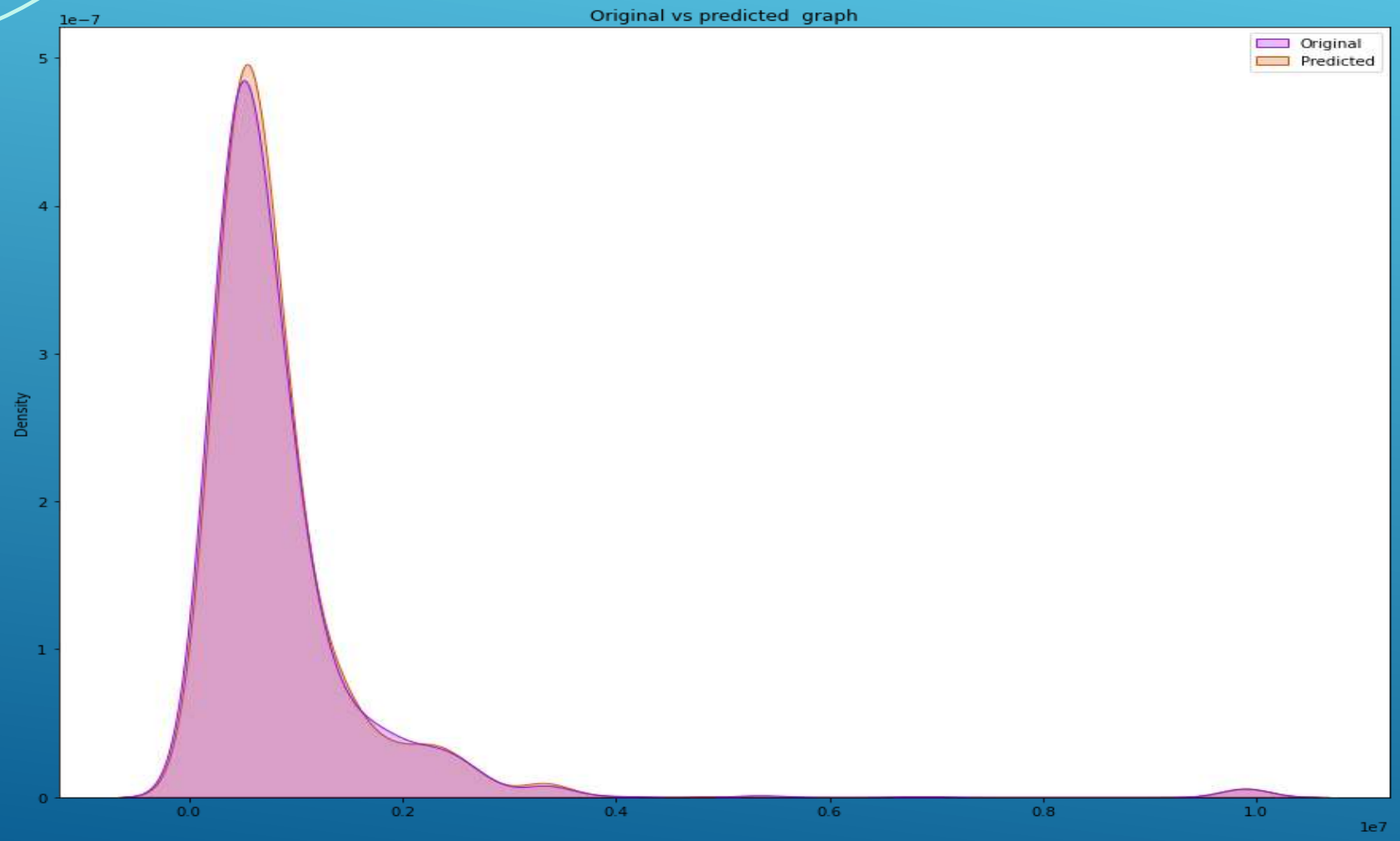
2. COMPARING ACTUAL AND PREDICTION

```
a = np.array(y_test)
predict = np.array(loaded_model.predict(x_test))
Car_price_prediction = pd.DataFrame({"Original":a,"Predicted":predict},index= range(len(a)))
Car_price_prediction
```

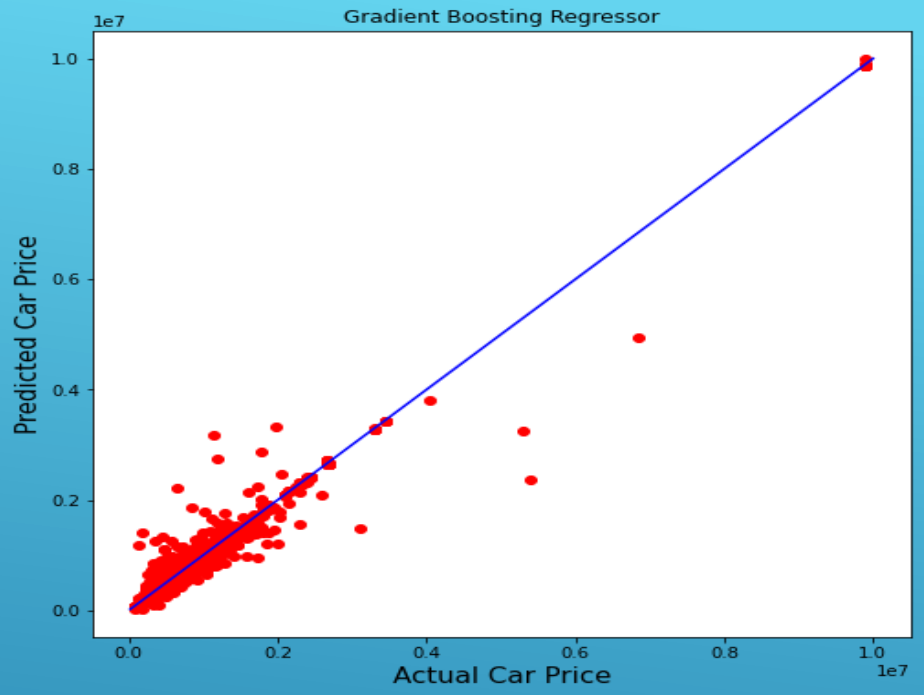
	Original	Predicted
0	265000.0	269749.292892
1	1084099.0	987743.584424
2	250000.0	248275.277621
3	603699.0	484027.938572
4	500000.0	485828.152206

Let's plot and visualize

```
plt.figure(figsize=(15,12))
sns.kdeplot(data=Car_price_prediction, palette='gnuplot',gridsize=900, shade=True)
plt.title('Original vs predicted  graph')
```



```
plt.figure(figsize=(8,8))
plt.scatter(y_test,predict,c='r')
plt1 = max(max(predict),max(y_test))
plt2 = min(min(predict),min(y_test))
plt.plot([plt1,plt2],[plt1,plt2],'b-')
plt.xlabel('Actual Car Price',fontsize=15)
plt.ylabel('Predicted Car Price',fontsize=15)
plt.title("Gradient Boosting Regressor")
plt.show()
```



3. SAVING THE MODEL IN CSV FORMAT

Saving the model in CSV format

```
model =Micro_Credit_Defaulter_Model.to_csv('Micro_Credit_Defaulter_Model.csv')
model
```

SUMMARY

- Here we have made a new car price valuation model as due to covid 19 impact previous car price valuation machine learning models is not working well because some cars are in demand hence making them costly and some are not in demand hence cheaper.

- For new car price valuation model, we have done prediction on basis of Data using EDA, Data Cleaning, Data Visualization, Data Pre-processing, Checked Correlation, removed irrelevant features , Removed Outliers, Removed Skewness and at last train our data by splitting our data through train-test split process.

- Built our model using 5 models and finally selected best model which was giving best accuracy that is Gradient Boosting Regressor. Then tunned our model through Hyper Tunning using GridSearchCV. And at last compared our predicted and Actual Price of Car. Thus our project is completed.

THANK YOU