



## Business Problem

Used Car Lot TM is a car sales company running for 2 years as a local business in India. The General Manager has requested some information about relative purchases of second hand vehicle in the regions. With fuel prices increasing and transport remaining to be a dominant necessity, some important business questions were presented for review.

1. What regions in India sold the highest number of used cars?
2. Does the fuel type affect the car price?
3. Elaborate on electric cars and do they make an impact on sales? (Elaborate from a business perspective).
4. What are some other factors that share a connection to sales prices?

From this analysis we intend on building a multiple linear regression (OLS) to model the relationships between the independent price and the dependent remaining variables in this dataset. The Used Car Lot intends on using the insights extracted to ensure their business prosperity and overall sales objectives are met proportionally. The information obtained will also be used to educate their sales team and help the business thrive.

## Step 1: Read the dataset and inspect its contents

```
In [1]: #Import packages
import numpy as np
import pandas as pd
import statsmodels.api as sm
import scipy.stats as stats
from matplotlib import pyplot as plt
from scipy import stats
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
import random
from math import sqrt
import seaborn as sns
import warnings
from scipy.stats import norm
warnings.filterwarnings("ignore")
%matplotlib inline
plt.style.use('seaborn')
```

```
In [2]: #Import the necessary libraries.
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
#from sklearn.model_selection import train_test_split
#from sklearn.Linear_model import LinearRegression
```

```
In [3]: #REview the dataset
df = pd.read_csv('data\\used_cars_data.csv')
```

In [4]: `#Load the first rows  
df.head()`

Out[4]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	M
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	

In [5]: `#dropping the first column  
df.drop(['S.No.'],axis=1,inplace=True)`

In [6]: `#dropping the S.No column  
df.drop(["New_Price"],axis=1,inplace=True)`

In [7]: `#specify data columns with null values to remove  
from scipy.stats import norm  
num_col = df.select_dtypes(include=np.number).columns.tolist()  
  
for col in num_col:  
 df[col]=df[col].replace(0.0,np.nan)`

In [8]: `#list categorical columns  
num_col`

Out[8]: `['Year', 'Kilometers_Driven', 'Seats', 'Price']`

```
In [9]: #the loop will add all the columns we want to change obj to string
# to use for a list
num_values = []

for colname in df.columns[df.dtypes == 'object']: # only need to focus on strings
    if df[colname].str.endswith('pl').any() or df[colname].str.endswith('kg').any():
        # using `str` to use an element-wise string method to select the required
        num_values.append(colname)

print(num_values)
['Mileage', 'Engine', 'Power']
```

```
In [10]: #Formulating a function to split string from the numerical values in the columns
## also drops the string and converts to float (see data head for values)
def obj_to_num(n):
    if isinstance(n,str): #checks if the column is a string
        if n.endswith('kmpL'):
            return float(n.split('kmpL')[0])
        elif n.endswith('km/kg'):
            return float(n.split('km/kg')[0])
        elif n.endswith('CC'):
            return float(n.split('CC')[0])
        elif n.startswith('null'):      #changes str values from 'null bhp' to Nan
            return np.nan
        elif n.endswith('bhp'):
            return float(n.split('bhp')[0])
    else:
        return np.nan

for colname in num_values:
    df[colname] = df[colname].apply(obj_to_num)#applying above function to the column
    df[colname]=df[colname].replace(0.0,np.nan)
```

```
In [11]: #Creating category
df["Name"] = df["Name"].astype("category")
df["Location"] = df["Location"].astype("category")
df["Fuel_Type"] = df["Fuel_Type"].astype("category")
df["Transmission"] = df["Transmission"].astype("category")
df["Owner_Type"] = df["Owner_Type"].astype("category")
```

In [12]: `#checking dataset for non-null values, types, totals  
df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name              7253 non-null   category
 1   Location          7253 non-null   category
 2   Year              7253 non-null   int64   
 3   Kilometers_Driven 7253 non-null   int64   
 4   Fuel_Type          7253 non-null   category
 5   Transmission       7253 non-null   category
 6   Owner_Type         7253 non-null   category
 7   Mileage            7170 non-null   float64 
 8   Engine             7207 non-null   float64 
 9   Power              7078 non-null   float64 
 10  Seats              7199 non-null   float64 
 11  Price              6019 non-null   float64 
dtypes: category(5), float64(5), int64(2)
memory usage: 520.7 KB
```

In [13]: `#creating dataframe  
df.describe(include=["category"]).T`

Out[13]:

	count	unique	top	freq
<b>Name</b>	7253	2041	Mahindra XUV500 W8 2WD	55
<b>Location</b>	7253	11	Mumbai	949
<b>Fuel_Type</b>	7253	5	Diesel	3852
<b>Transmission</b>	7253	2	Manual	5204
<b>Owner_Type</b>	7253	4	First	5952

Mumbai sold the most cars Preference of fuel choice is diesel Manual transmission sold most First time owners purchased a second hand car the most in this dataset

In [14]: `#Split brand and the car model  
df[['Car_Brand', 'Model']] = df.Name.str.split(n=1,expand=True)`

In [15]: `# Model names are unique to the Car Brands  
Brand_name=df['Car_Brand'].unique()  
Model=df['Model'].unique()`

```
In [16]: #Revising names
df['Car_Brand']=df['Car_Brand'].replace('Land','Land_Rover')
df['Car_Brand']=df['Car_Brand'].replace('ISUZU','Isuzu')
df['Car_Brand'].value_counts()
```

```
Out[16]: Maruti          1444
Hyundai         1340
Honda            743
Toyota           507
Mercedes-Benz    380
Volkswagen       374
Ford              351
Mahindra          331
BMW              312
Audi              285
Tata              228
Skoda             202
Renault            170
Chevrolet          151
Nissan             117
Land_Rover          67
Jaguar              48
Fiat                38
Mitsubishi          36
Mini                31
Volvo               28
Porsche              19
Jeep                 19
Datsun              17
Isuzu                5
Force                3
Bentley              2
Smart                1
Ambassador            1
Lamborghini            1
Hindustan              1
OpelCorsa              1
Name: Car_Brand, dtype: int64
```

```
In [17]: #creating num columns
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()
numeric_columns
```

```
Out[17]: ['Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power', 'Seats', 'Price']
```

```
In [18]: #creating dependant variable using Lambda function
numeric_columns.remove('Price')
medianFiller = lambda x: x.fillna(x.median())
df[numeric_columns] = df[numeric_columns].apply(medianFiller, axis=0)
```

```
In [19]: #creating List for column data
Median1=[]
for i in range(len(brand_name)):
    x=df['Price'][df['Car_Brand']==brand_name[i]].median()
    Median1.append(x)
```

```
In [20]: #Creating a second empty list to display median price of cars per Car model for v
Median2=[]
for i in range(len(Model)):
    x=df['Price'][df['Model']==Model[i]].median()
    Median2.append(x)
```

```
In [21]: #adding float amount 0.00 to missing data values
df['Price']= df['Price'].fillna(0.0)
```

```
In [22]: #running a loop to check every row in df dataset
for i in range(len(df)):
    if df.Price[i]==0.00:
        for j in range(len(Model)):
            if df.Model[i]==Model[j]: #Comparing the Car model names in both datasets
                df.Price[i]=Median2[j] #replacing the Price of the car with the median value
```

```
In [23]: #re-checking for types and confirming non-null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name             7253 non-null   category
 1   Location         7253 non-null   category
 2   Year             7253 non-null   int64   
 3   Kilometers_Driven 7253 non-null   int64   
 4   Fuel_Type         7253 non-null   category
 5   Transmission     7253 non-null   category
 6   Owner_Type       7253 non-null   category
 7   Mileage          7253 non-null   float64 
 8   Engine            7253 non-null   float64 
 9   Power             7253 non-null   float64 
 10  Seats             7253 non-null   float64 
 11  Price             7075 non-null   float64 
 12  Car_Brand        7253 non-null   object  
 13  Model             7253 non-null   object  
dtypes: category(5), float64(5), int64(2), object(2)
memory usage: 634.0+ KB
```

In [24]: #checking for all price values containing NaN's in df  
df[df['Price'].isna()]

Out[24]:

		Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
6021		Toyota Innova Crysta Touring Sport 2.4 MT	Mumbai	2017	34000	Diesel	Manual	First	13.60
6037		Maruti Swift AMT ZXI	Kochi	2019	15409	Petrol	Automatic	First	22.00
6042		Skoda Laura 1.8 TSI Ambition	Bangalore	2009	72000	Petrol	Manual	Second	17.50
6043		Honda Civic 2010- 2013 1.8 S MT Inspire	Bangalore	2011	40000	Petrol	Manual	Second	15.50
6076		Toyota Etios Liva 1.4 VXD	Pune	2016	104350	Diesel	Manual	First	23.50
...	...	...	...	...	...	...	...	...	...
7225		Skoda Superb Petrol Ambition	Delhi	2010	40000	Petrol	Manual	First	13.10
7228		Tata Tiago 1.05 Revotorq XT Option	Pune	2016	41413	Diesel	Manual	First	27.20
7231		Ford EcoSport 1.5 Petrol Ambiente	Kochi	2017	39015	Petrol	Manual	First	17.00
7232		Jeep Compass 1.4 Sport	Mumbai	2018	6000	Petrol	Manual	First	16.00
7236		Hyundai Elite i20 Magna Plus	Kochi	2018	23955	Petrol	Manual	First	18.60

178 rows × 14 columns

```
In [25]: #Again, replacing the missing values with float 0.0
df['Price']= df['Price'].fillna(0.0)
#running a loop to check row in dataset
for i in range(len(df)):
    if df.Price[i]==0.00:
        for j in range(len(Brand_name)):
            if df.Car_Brand[i]==Brand_name[j]: #Comparing the brand names in both
                df.Price[i]=Median1[j]      #replacing with corresponding median value
```

```
In [26]: #Again, reviewing NaN Price values
df[df['Price'].isna()]
```

Out[26]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
6216	Hindustan Motors Contessa 2.0 DSL	Pune	1996	65000	Diesel	Manual	Second	14
7067	OpelCorsa 1.4Gsi	Hyderabad	2005	50000	Petrol	Manual	Second	14

```
In [27]: #Dropping the above cars - as they are rare and will not impact results
df.dropna(axis=0,inplace=True)
```

```
In [28]: #checking the shape
df.shape
```

Out[28]: (7251, 14)

```
In [29]: #Lambda function for float values and transpose data frame output
pd.set_option('display.float_format', lambda x: '%.3f' % x)
df.describe().T
```

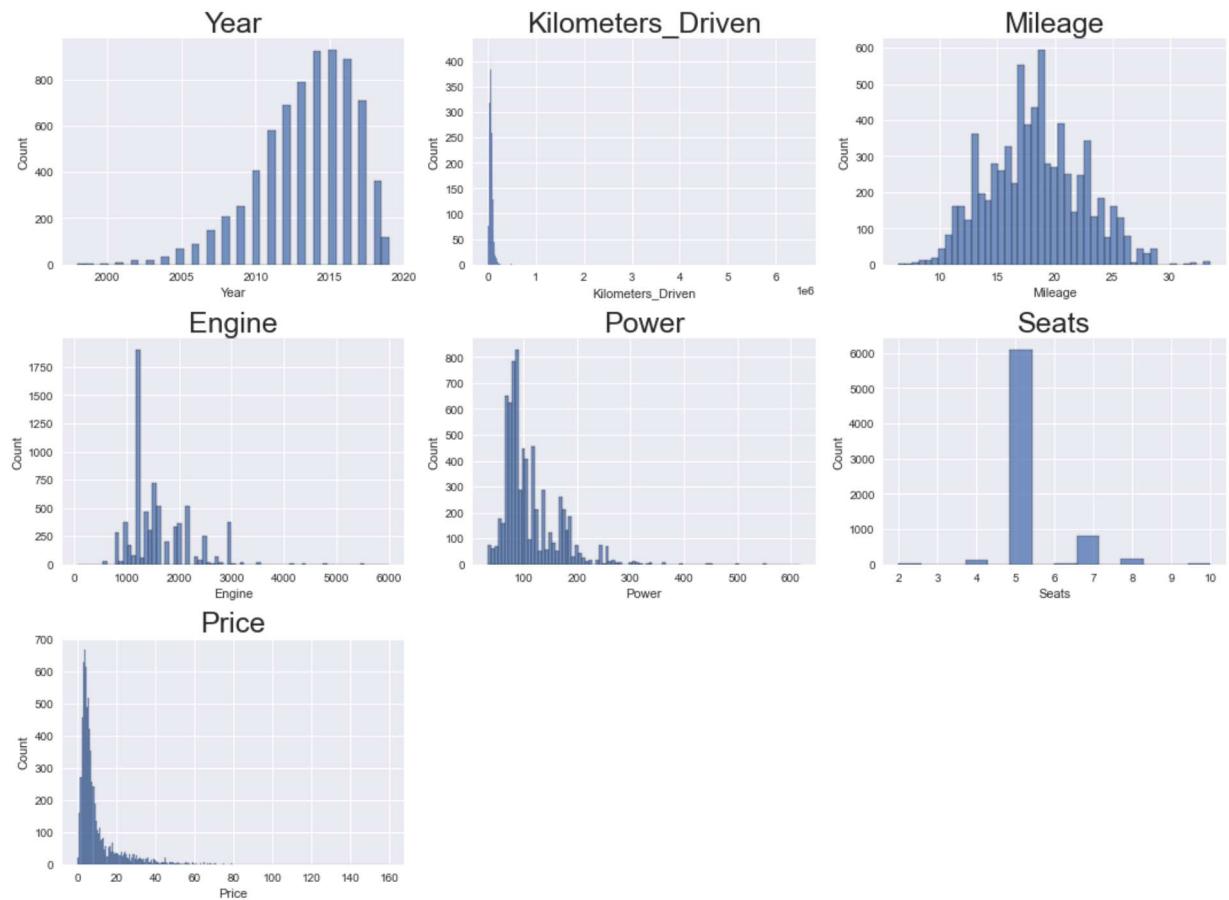
Out[29]:

	count	mean	std	min	25%	50%	75%	max
Year	7251.000	2013.369	3.247	1998.000	2011.000	2014.000	2016.000	2019.000
Kilometers_Driven	7251.000	58699.394	84439.271	171.000	34000.000	53416.000	73000.000	65000.000
Mileage	7251.000	18.346	4.134	6.400	15.400	18.200	21.100	25.000
Engine	7251.000	1615.769	593.534	72.000	1198.000	1493.000	1968.000	2200.000
Power	7251.000	112.318	52.929	34.200	77.000	94.000	138.030	180.000
Seats	7251.000	5.278	0.807	2.000	5.000	5.000	5.000	5.000
Price	7251.000	9.328	10.863	0.000	3.500	5.500	9.750	14.000

## Plot histograms to check the distributions of the predictors

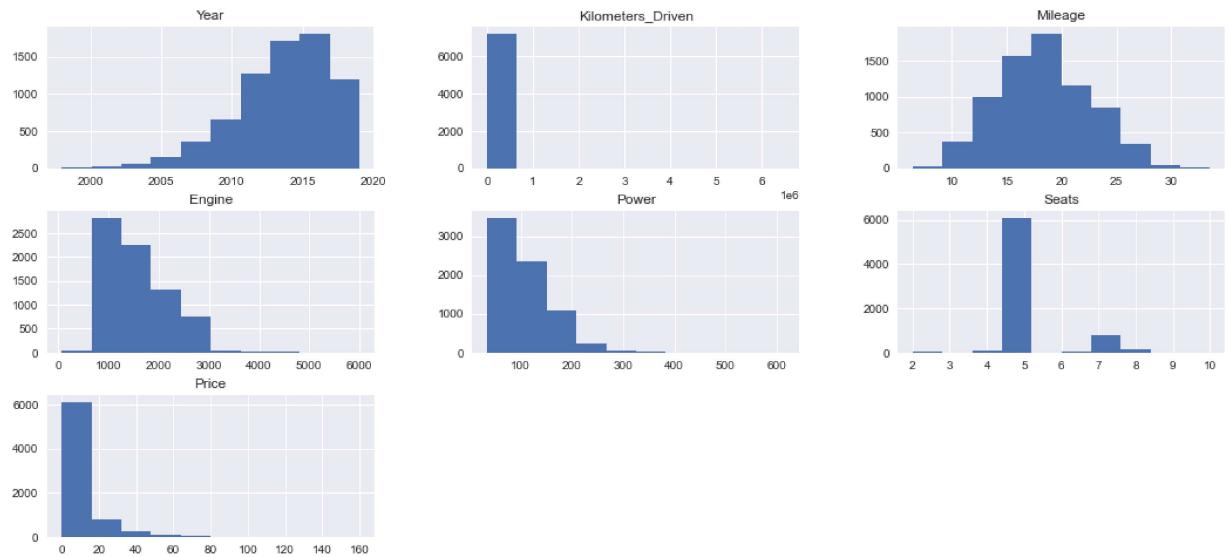
```
In [30]: #Plotting histogram for distribution
Uni_num = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(15,65))
#creating a loop that will show the plots for the columns in one plot
for i in range(len(Uni_num)):
    plt.subplot(18,3,i+1)
    sns.histplot(df[Uni_num[i]],kde=False)
    plt.tight_layout()
    plt.title(Uni_num[i],fontsize=25)

plt.show()
```



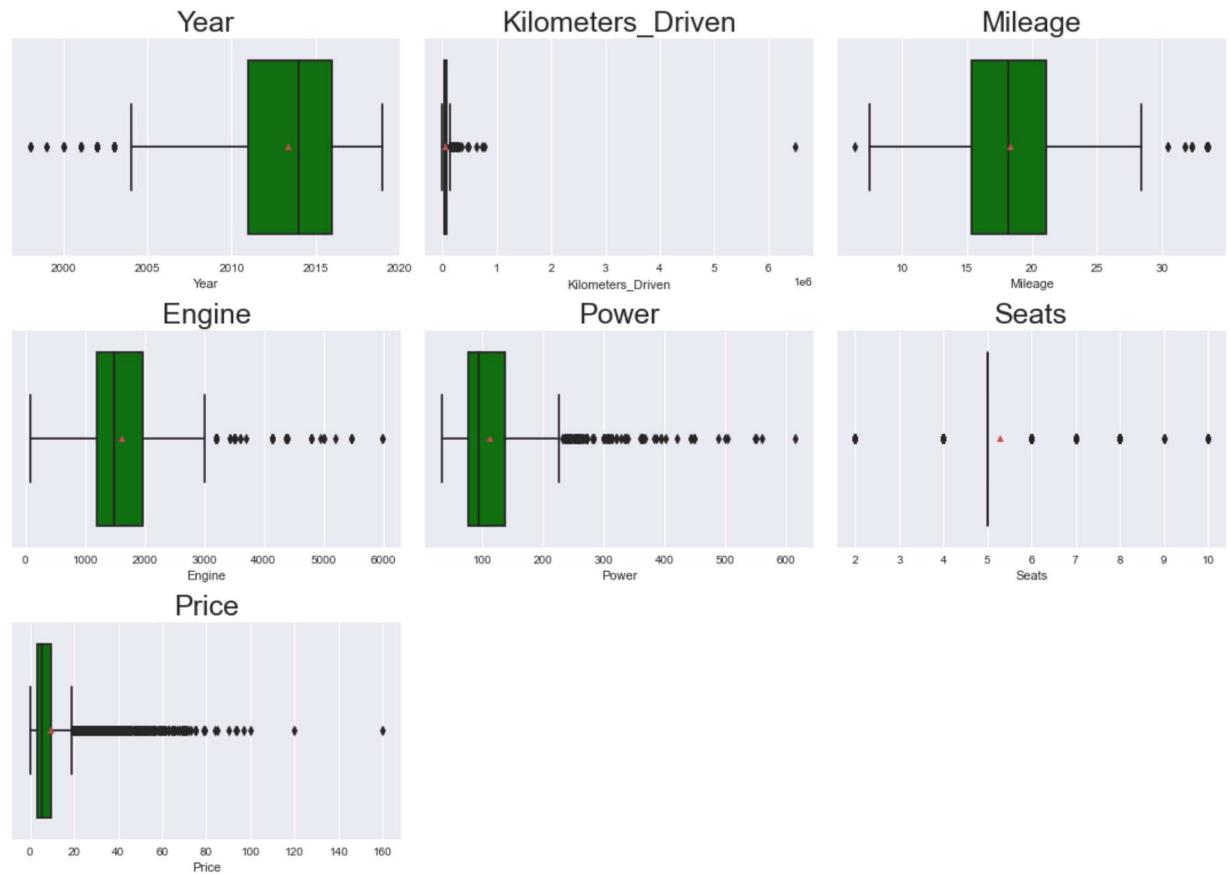
Analysis of above and the below visualisations  
 Year: Left Skewed  
 Kilometers driven: Right skewed  
 Mileage: Normal distribution  
 Engine: Right skewed  
 Power: Right skewed  
 Seats: Right Skewed

```
In [31]: #Useful histogram to view any variances and understand the data set - note mileage
import warnings
warnings.filterwarnings('ignore')
fig = plt.figure(figsize = (18,8))
ax = fig.gca()
df.hist(ax = ax);
```



In [32]: *#Plotting a box plot to investigate parameters*

```
plt.figure(figsize=(15,35))
for i in range(len(Uni_num)):
    plt.subplot(10,3,i+1)
    sns.boxplot(df[Uni_num[i]],showmeans=True, color='green')
    plt.tight_layout()
    plt.title(Uni_num[i],fontsize=25)
plt.show()
```



In [33]: #Creating region lists

```
regions ={'Delhi':'North','Jaipur':'North',
          'Chennai':'South','Coimbatore':'South','Hyderabad':'South','Bangalore':'South',
          'Kolkata':'East',
          'Mumbai':'West','Pune':'West','Ahmedabad':'West'}
df['Region']=df['Location'].replace(regions)
```

In [34]: #dropping unnecessary models & counting valid car type values to classify by Level

```
df.drop(["Car_Brand","Model"],axis=1,inplace=True)
df['Car_Type'] = pd.cut(df['Price'],[-np.inf,5.5,10.5,20.5,45.0,75.0,np.inf],
                        labels=["Level1","Level2","Level3","Level4","Level5","Level6"])
df['Car_Type'].value_counts()

## LEVELS = CAR_TYPE Level 1 is Low grade(economical less expensive car) to Level6
## Owner Type = First second cars, second card, third car, etc.
```

Out[34]:

Level1	3638
Level2	1916
Level3	896
Level4	669
Level5	120
Level6	12
Name:	Car_Type, dtype: int64

In [35]: #View sample of Level 3 type vehicle below

```
df.sample()
```

Out[35]:

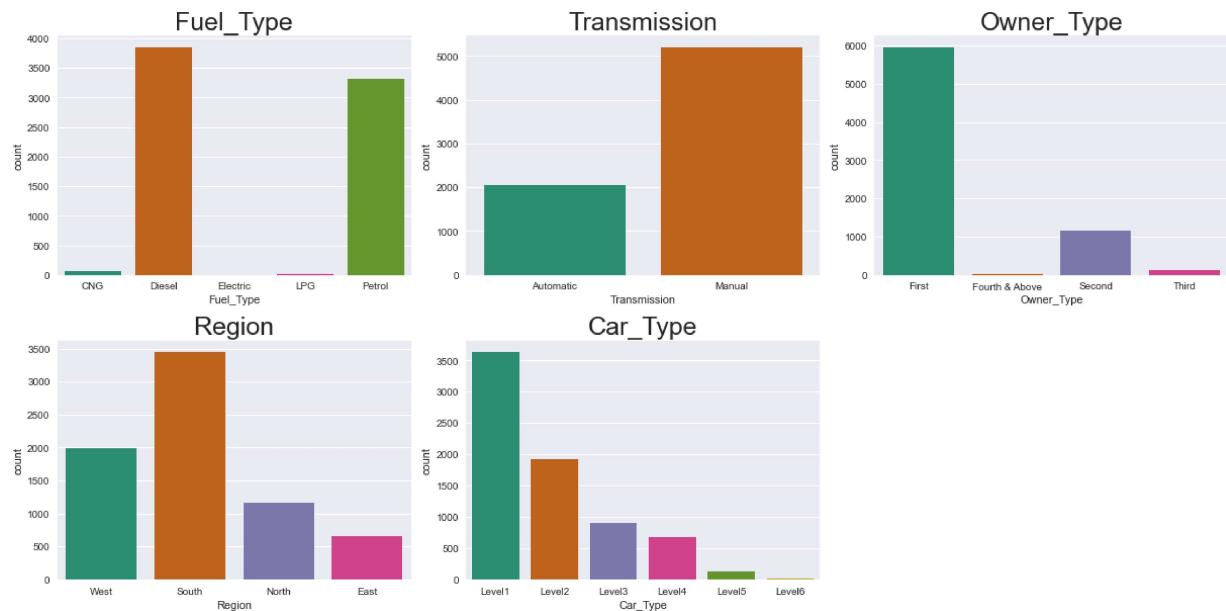
		Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
		Ford Figo 2015- 3576	Hyderabad	2016 1.5D Titanium MT	71000	Diesel	Manual	First	25.830



```
In [36]: #set categorical values to craete column list
categorical_val = df.select_dtypes(exclude=np.number).columns.tolist()
categorical_val.remove('Name')
categorical_val.remove('Location')
```

In [37]: #visual bar graph

```
plt.figure(figsize=(17,75))
for i in range(len(categorical_val)):
    plt.subplot(18,3,i+1)
    ax=sns.countplot(df[categorical_val[i]],palette='Dark2')
    plt.tight_layout()
    plt.title(categorical_val[i],fontsize=25)
    total = len(df[categorical_val[i]])
    for p in ax.patches:
        x = p.get_x() + (p.get_width() / 2)-0.1
        y = p.get_y() + p.get_height()
plt.show()
```



In [38]: #Reviewing dataset for electric cars

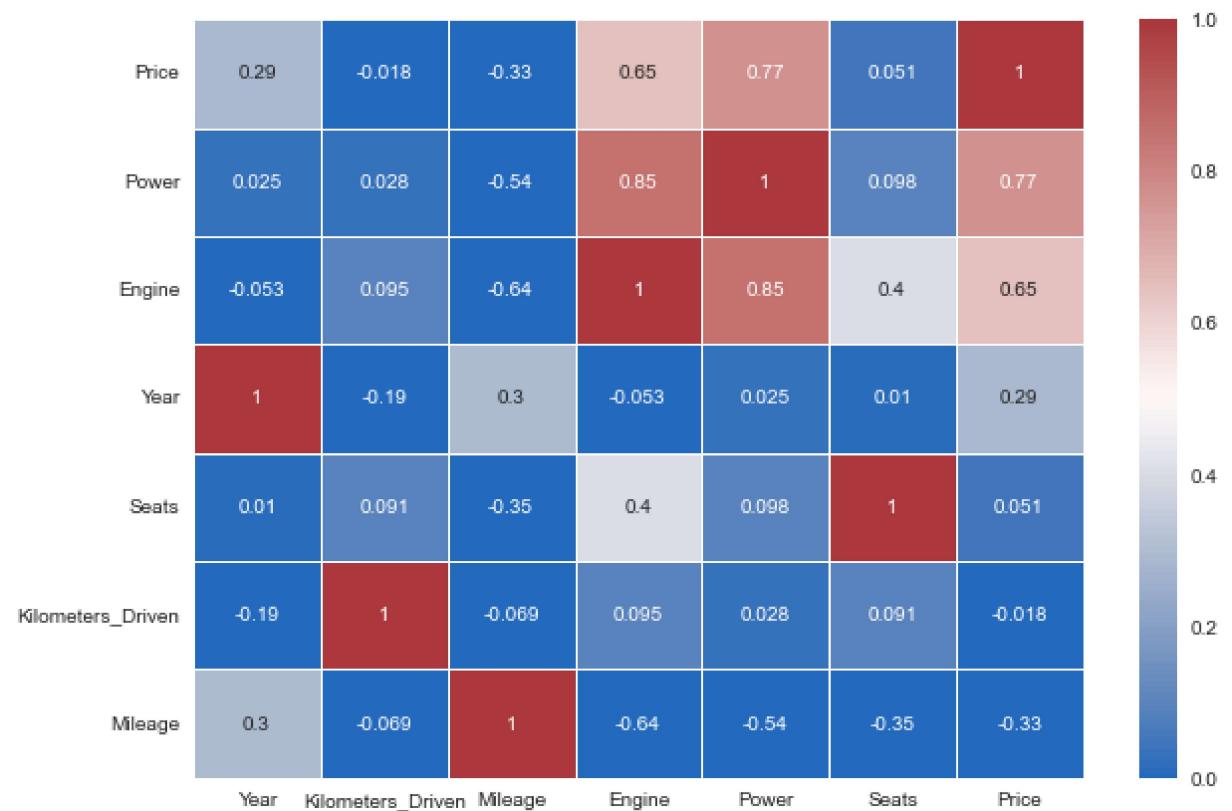
```
df[df['Fuel_Type']=='Electric']
```

Out[38]:

		Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
4446		Mahindra E Verito D4	Chennai	2016	50000	Electric	Automatic	First	18.200
4904		Toyota Prius 2009-2016 Z4	Mumbai	2011	44000	Electric	Automatic	First	18.200

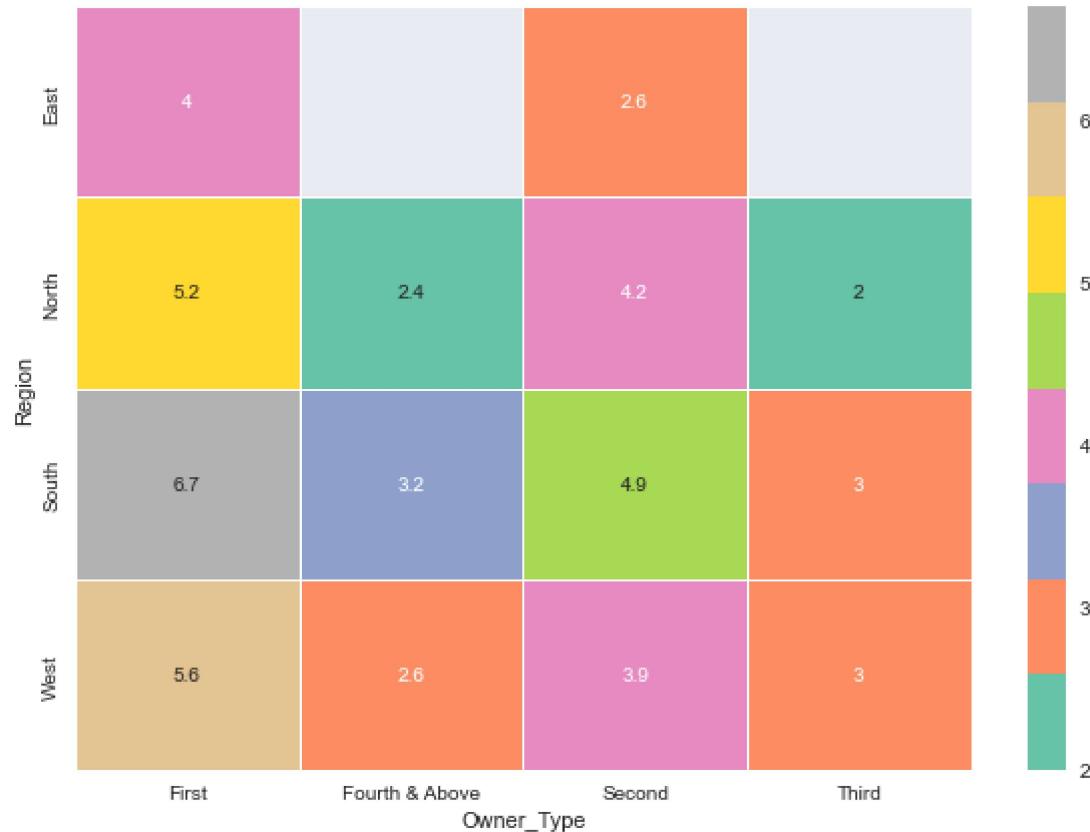
In [39]: *#heat map showing strongest correlations*

```
corr= df.corr().sort_values(by=['Price'],ascending=False)
plt.figure(figsize=(10,7))
sns.heatmap(corr,annot= True,vmin=0,vmax=1, cmap='vlag',linewdiths=0.75)
plt.show()
#Price & Power = .77
#Power & Engine = .85
```



In [40]: #Does type of ownership affect Car price?

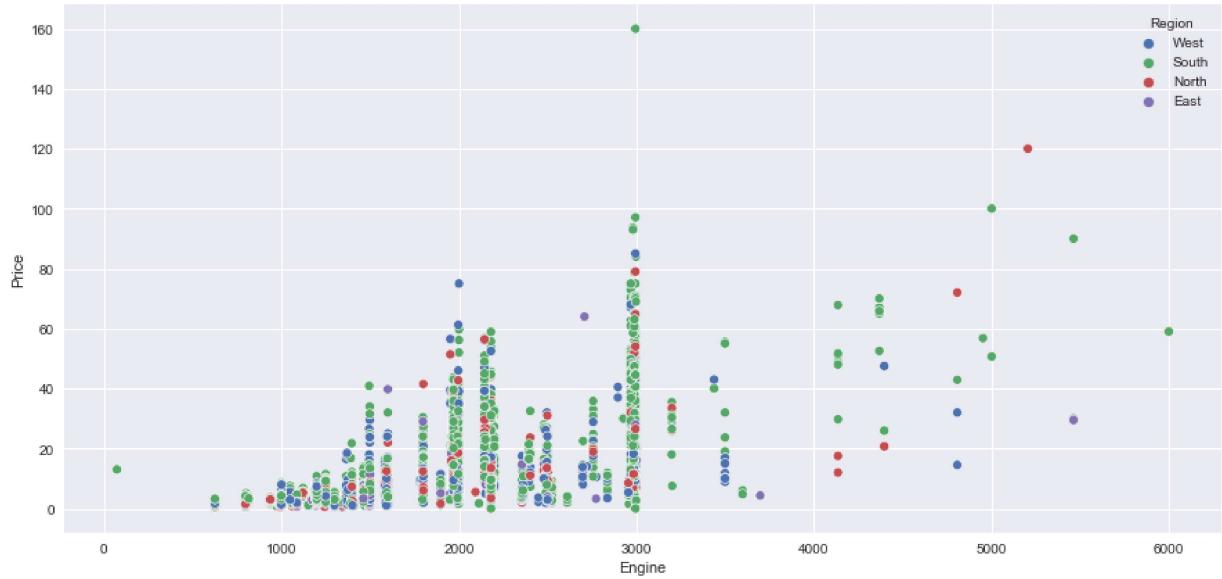
```
df_hm = df.pivot_table(index = 'Region',columns ='Owner_Type',values ="Price",aggfunc='mean')
# Heatmap to display
plt.subplots(figsize=(10,7))
sns.heatmap(df_hm,cmap='Set2',linewidths=.5, annot=True);
#south Region purchases more second hand cars
```



### Step 3: Test for the linearity assumption

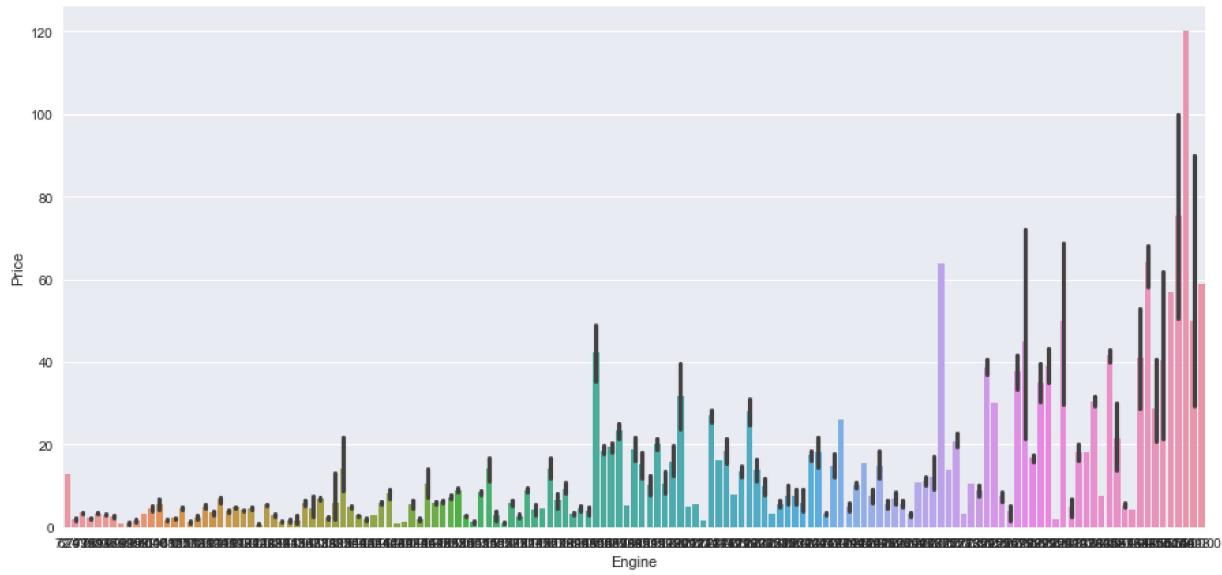
Using scatterplots, heatmaps, bar graphs to plot each predictor against the target variable

```
In [41]: #Analysis of variables that have high correlation with Price
#Price Vs Engine Vs Region
plt.figure(figsize=(15,7))
sns.scatterplot(data=df,y='Price',x='Engine',hue='Region')
plt.show()
```



```
In [42]: #Does type of Fuel affect car price?
plt.figure(figsize=(15,7))
sns.barplot(data=df,x='Engine',y='Price')
```

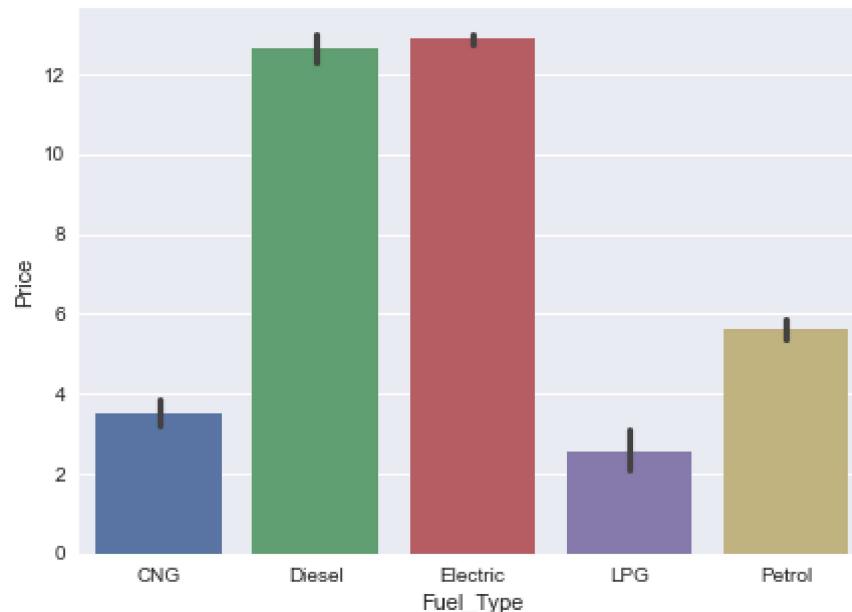
Out[42]: <AxesSubplot:xlabel='Engine', ylabel='Price'>



In [43]: #Does type of Fuel affect car price?

```
plt.figure(figsize=(7,5))  
sns.barplot(data=df,x='Fuel_Type',y='Price')
```

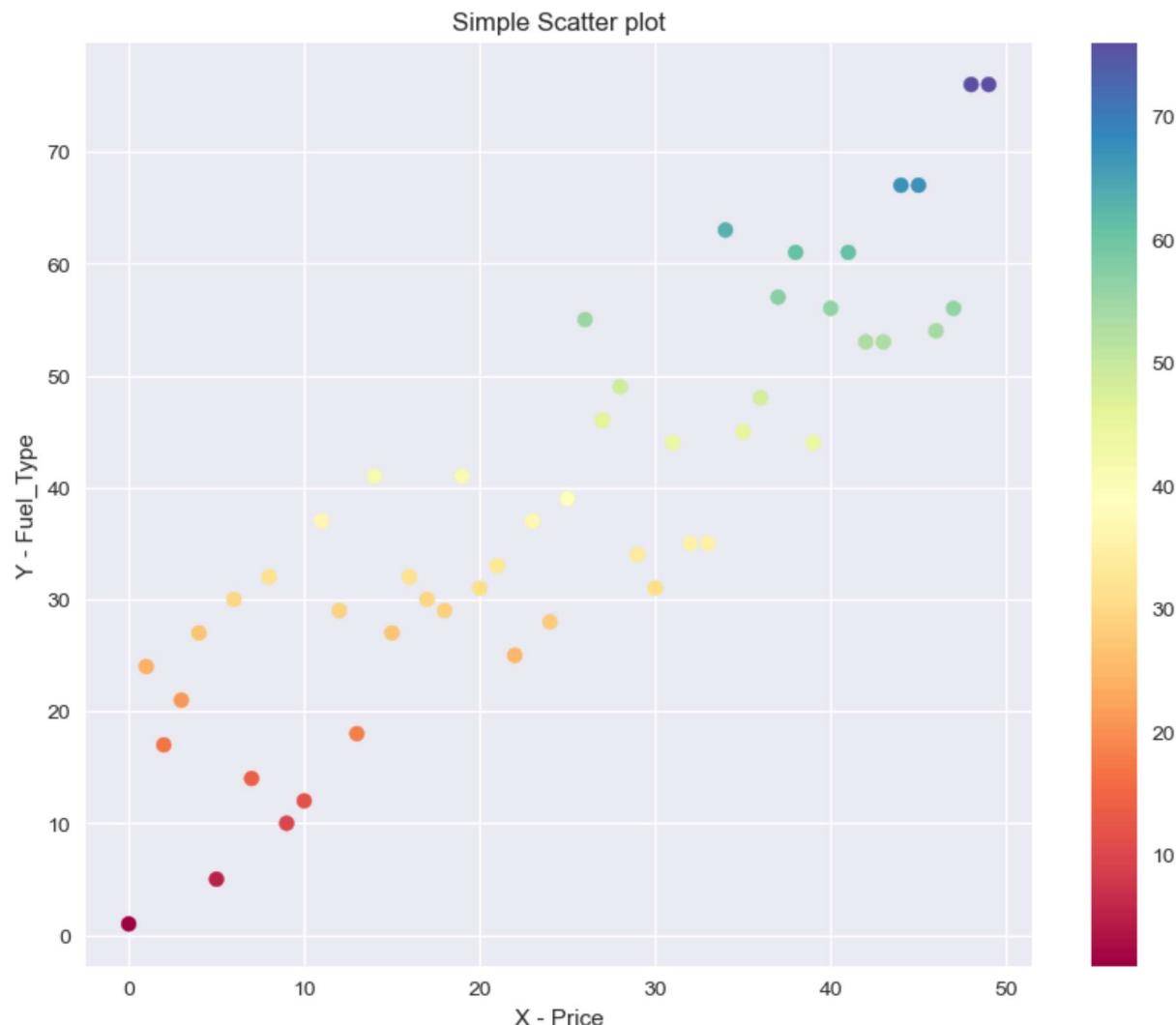
Out[43]: <AxesSubplot:xlabel='Fuel\_Type', ylabel='Price'>



## Step 7: Visualize the error term for variance and heteroscedasticity

In [44]: # Simple Scatterplot with colored points

```
x = range(50)
y = range(50) + np.random.randint(0,30,50)
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
plt.scatter(x, y, c=y, cmap='Spectral')
plt.colorbar()
plt.title('Simple Scatter plot')
plt.xlabel('X - Price')
plt.ylabel('Y - Fuel_Type')
plt.show()
```



```
In [45]: #Dealing with outliers utilising the 25th & 75th quantiles
def treat_outliers(df,col):
```

```
    Q1=df[col].quantile(0.25) # 25th quantile
    Q3=df[col].quantile(0.75) # 75th quantile
    IQR=Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker)

    return df
```

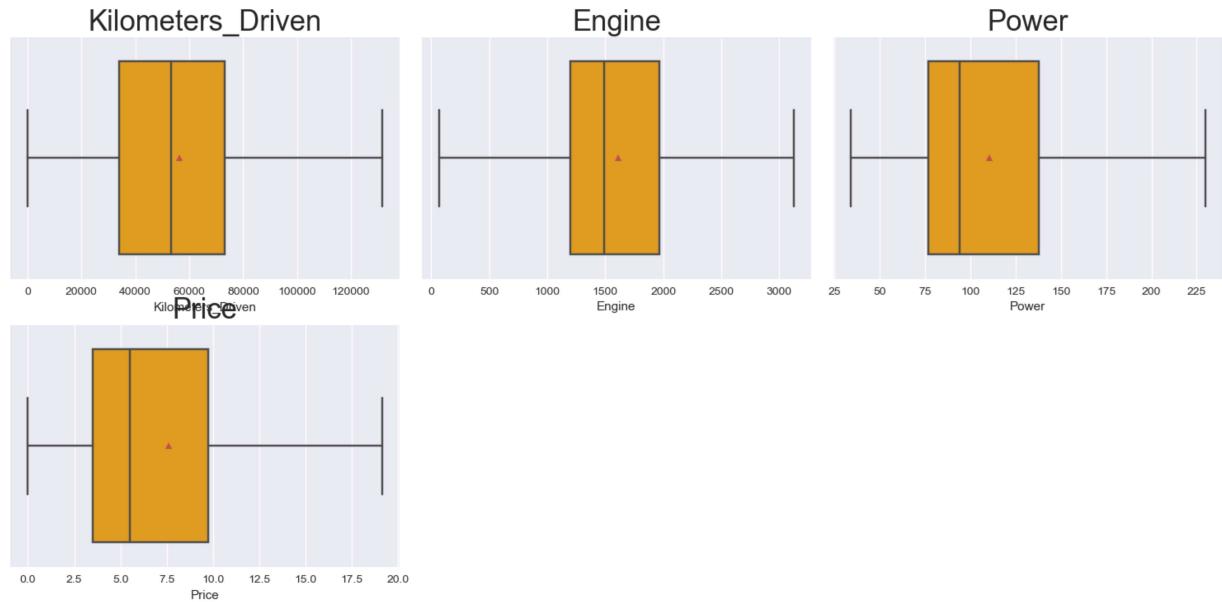
```
def treat_outliers_all(df, col_list):
    for c in col_list:
        df = treat_outliers(df,c)
    return df
```

```
In [46]: #making a first copy
#Dropping Year,Mileage and Seats irrelevant due to outliers
df2=df.copy()
numerical_col = df2.select_dtypes(include=np.number).columns.tolist()
numerical_col.remove('Year')
numerical_col.remove('Mileage')
numerical_col.remove('Seats')
numerical_col
```

```
Out[46]: ['Kilometers_Driven', 'Engine', 'Power', 'Price']
```

```
In [47]: #dealing with outliers
df2 = treat_outliers_all(df2,numerical_col)
```

```
In [48]: #checking if the outliers have applied
plt.figure(figsize=(15,35))
for i in range(len(numerical_col)):
    plt.subplot(10,3,i+1)
    sns.boxplot(df2[numerical_col[i]], showmeans=True, color='orange')
    plt.tight_layout()
    plt.title(numerical_col[i], fontsize=25)
plt.show()
```



```
In [49]: #Reviewing data head
df2.head()
```

Out[49]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000.000	CNG	Manual	First	26.600
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000.000	Diesel	Manual	First	19.670
2	Honda Jazz V	Chennai	2011	46000.000	Petrol	Manual	First	18.200
3	Maruti Ertiga VDI	Chennai	2012	87000.000	Diesel	Manual	First	20.770
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670.000	Diesel	Automatic	Second	15.200

```
In [50]: #dropping Name as we have created a bin called: Car_Type
#dropping Fuel_Type
X = df2.drop(['Name', 'Fuel_Type', 'Location', 'Price'], axis=1)
y = df2[['Price']]
#Reviewing new data shape
print(X.shape)
print(y.shape)
```

(7251, 10)  
(7251, 1)

```
In [51]: #Creating Dummy Variables for the Categorical Columns
#Dummy variable will be used as independent variables and will not impose any rare
X = pd.get_dummies(X, columns=['Transmission', 'Owner_Type', 'Region', 'Car_Type'],
X.head()
```

Out[51]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Transmission_Manual	Owner_Type_&
0	2010	72000.000	26.600	998.000	58.160	5.000		1
1	2015	41000.000	19.670	1582.000	126.200	5.000		1
2	2011	46000.000	18.200	1199.000	88.700	5.000		1
3	2012	87000.000	20.770	1248.000	88.760	7.000		1
4	2013	40670.000	15.200	1968.000	140.800	5.000		0

◀ ▶

```
In [52]: #split the data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
X_train.head()
```

◀ ▶

Out[52]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Transmission_Manual	Owner_Ty
1928	2016	55100.000	17.010	1582.000	126.200	5.000		0
6278	2012	90000.000	23.590	1364.000	94.000	5.000		1
22	2015	55985.000	13.530	1984.000	177.010	5.000		0
5925	2010	85000.000	17.500	1798.000	94.000	5.000		1
5762	2014	51000.000	27.030	1969.000	190.000	5.000		0

◀ ▶

```
In [53]: #Fitting Linear model
from sklearn.linear_model import LinearRegression
linearregression = LinearRegression()
linearregression.fit(X_train, y_train)
print("Intercept of the linear equation:", linearregression.intercept_)
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, linearregression.coef_[idx]))
```

Intercept of the linear equation: [-404.87654936]  
The coefficient for Year is 0.2014249816152108  
The coefficient for Kilometers\_Driven is -1.6203755332221522e-06  
The coefficient for Mileage is 0.04349748263039035  
The coefficient for Engine is 0.0007955011277483669  
The coefficient for Power is 0.016005440241528498  
The coefficient for Seats is 0.07091913468838472  
The coefficient for Transmission\_Manual is -0.7422961502300802  
The coefficient for Owner\_Type\_Fourth & Above is 0.5406481259194158  
The coefficient for Owner\_Type\_Second is -0.09782155395636072  
The coefficient for Owner\_Type\_Third is -0.23169983508213976  
The coefficient for Region\_North is 0.21245763647508623  
The coefficient for Region\_South is 0.3898811409975626  
The coefficient for Region\_West is 0.26965728280936585  
The coefficient for Car\_Type\_Level2 is 2.6033256086328653  
The coefficient for Car\_Type\_Level3 is 8.598665328915168  
The coefficient for Car\_Type\_Level4 is 12.04933176079081  
The coefficient for Car\_Type\_Level5 is 10.894805905199384  
The coefficient for Car\_Type\_Level6 is 10.584283476822167

```
In [54]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pred = linearregression.predict(X_test)
```

```
In [55]: # Mean Absolute Error on test
mean_absolute_error(y_test, pred)
```

Out[55]: 0.8903015607952957

```
In [56]: # RMSE on test data
mean_squared_error(y_test, pred)**0.5
```

Out[56]: 1.215184412886623

```
In [57]: # R-squared on test
r2_score(y_test, pred)
```

Out[57]: 0.9536193921069149

```
In [58]: #Calculating train score
linearregression.score(X_train, y_train) # 70 % data
```

Out[58]: 0.9520061630151408

```
In [59]: #Calculating Test Score
linearregression.score(X_test, y_test) # unseen data
```

Out[59]: 0.9536193921069149

In [60]: #Creating another copy of df

```
df3=df.copy()
df3.head()
```

Out[60]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.600
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.670
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.200
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.770
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.200

◀ ▶

In [61]: #Creating additional variable to drop unnecessary columns

```
X1 = df3.drop(['Name', 'Fuel_Type', 'Location', 'Price'], axis=1)
y1 = df3[['Price']]
```

```
print(X1.shape)
print(y1.shape)
```

```
(7251, 10)
(7251, 1)
```

In [62]: #Creating Dummy Variables for categorical columns

```
X1 = pd.get_dummies(X1, columns=['Region', 'Car_Type', 'Transmission', 'Owner_Type'])
X1.head()
```

Out[62]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Region_North	Region_South	Regic
0	2010	72000	26.600	998.000	58.160	5.000	0	0	
1	2015	41000	19.670	1582.000	126.200	5.000	0	0	
2	2011	46000	18.200	1199.000	88.700	5.000	0	1	
3	2012	87000	20.770	1248.000	88.760	7.000	0	1	
4	2013	40670	15.200	1968.000	140.800	5.000	0	1	

◀ ▶

```
In [63]: #split the data into train and test
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.3, random_state=42)
X1_train.head()
```

Out[63]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Region_North	Region_South	R
1928	2016	55100	17.010	1582.000	126.200	5.000	0	0	0
6278	2012	90000	23.590	1364.000	94.000	5.000	0	1	0
22	2015	55985	13.530	1984.000	177.010	5.000	0	0	0
5925	2010	85000	17.500	1798.000	94.000	5.000	0	0	0
5762	2014	51000	27.030	1969.000	190.000	5.000	0	1	0

```
In [64]: #Fitting Linear model
from sklearn.linear_model import LinearRegression
linearregression = LinearRegression()
linearregression.fit(X1_train, y1_train)
print('Intercept of the linear equation:', linearregression.intercept_)
for idx, col_name in enumerate(X1_train.columns):
    print("The coefficient for {} is {}".format(col_name, linearregression.coef_[idx]))
```

Intercept of the linear equation: [-559.64477119]  
The coefficient for Year is 0.27815207711649104  
The coefficient for Kilometers\_Driven is -3.933049214243794e-06  
The coefficient for Mileage is 0.041173020980351936  
The coefficient for Engine is 0.0012536524615097344  
The coefficient for Power is 0.021857197060797674  
The coefficient for Seats is -0.10443213439518366  
The coefficient for Region\_North is 0.43423501867368897  
The coefficient for Region\_South is 0.6551430985061976  
The coefficient for Region\_West is 0.3849291968220981  
The coefficient for Car\_Type\_Level2 is 2.18734590987969  
The coefficient for Car\_Type\_Level3 is 7.852889631704657  
The coefficient for Car\_Type\_Level4 is 21.448157673682097  
The coefficient for Car\_Type\_Level5 is 45.73101094812465  
The coefficient for Car\_Type\_Level6 is 77.34107051395367  
The coefficient for Transmission\_Manual is -0.5553857640064356  
The coefficient for Owner\_Type\_Fourth & Above is 1.060293629823952  
The coefficient for Owner\_Type\_Second is -0.012176133022344845  
The coefficient for Owner\_Type\_Third is 0.19972117592182498

```
In [65]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pred1 = linearregression.predict(X1_test)
```

```
In [66]: # Calculate MAE
mean_absolute_error(y1_test, pred1)
```

Out[66]: 1.5128627827869912

```
In [67]: #Calculate RMSE  
mean_squared_error(y1_test, pred1)**0.5
```

```
Out[67]: 3.093684570463301
```

```
In [68]: #Calculate R2 Score  
r2_score(y1_test, pred1)
```

```
Out[68]: 0.921506918435542
```

```
In [69]: #Calculate training Lr Score based on 70% training data  
linearregression.score(X1_train, y1_train)
```

```
Out[69]: 0.9450553498559194
```

The above appears to be a good fit to model predictions at 94%

```
In [70]: #calculate lr score  
linearregression.score(X1_test, y1_test)
```

```
Out[70]: 0.921506918435542
```

## Run a OLS regression in Statsmodels with multiple variables in test data

```
In [71]: # Builing first OLS stats model
import statsmodels.api as sm
X = sm.add_constant(X)
X_train1, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
         _state=42)

olsmod0 = sm.OLS(y_train, X_train1)
olsres0 = olsmod0.fit()
print(olsres0.summary())
```

### OLS Regression Results

```
=====
Dep. Variable: Price R-squared: 0.95
Model: OLS Adj. R-squared: 0.95
Method: Least Squares F-statistic: 557
Date: Sat, 11 Jun 2022 Prob (F-statistic): 0.0
Time: 18:55:52 Log-Likelihood: -8236.
No. Observations: 5075 AIC: 1.651e+0
Df Residuals: 5056 BIC: 1.664e+0
Df Model: 18
Covariance Type: nonrobust
=====
```

		coef	std err	t	P> t
[0.025	0.975]				
const		-404.8765	15.837	-25.565	0.000
5.924	-373.829				-43
Year		0.2014	0.008	25.530	0.000
0.186	0.217				
Kilometers_Driven		-1.62e-06	7.42e-07	-2.183	0.029
e-06	-1.65e-07				-3.08
Mileage		0.0435	0.006	7.141	0.000
0.032	0.055				
Engine		0.0008	7.79e-05	10.216	0.000
0.001	0.001				
Power		0.0160	0.001	16.028	0.000
0.014	0.018				
Seats		0.0709	0.029	2.463	0.014
0.014	0.127				
Transmission_Manual		-0.7423	0.056	-13.312	0.000
0.852	-0.633				-
Owner_Type_Fourth & Above		0.5406	0.412	1.311	0.190
0.268	1.349				-
Owner_Type_Second		-0.0978	0.051	-1.917	0.055
0.198	0.002				-
Owner_Type_Third		-0.2317	0.132	-1.754	0.079
0.491	0.027				-
Region_North		0.2125	0.074	2.869	0.004

0.067	0.358				
Region_South		0.3899	0.066	5.909	0.000
0.261	0.519				
Region_West		0.2697	0.068	3.940	0.000
0.135	0.404				
Car_Type_Level2		2.6033	0.050	51.616	0.000
2.504	2.702				
Car_Type_Level3		8.5987	0.079	108.186	0.000
8.443	8.754				
Car_Type_Level4		12.0493	0.104	116.153	0.000
1.846	12.253				1
Car_Type_Level5		10.8948	0.180	60.663	0.000
0.543	11.247				1
Car_Type_Level6		10.5843	0.411	25.764	0.000
9.779	11.390				
<hr/>					
=					
Omnibus:		358.893	Durbin-Watson:		1.99
8					
Prob(Omnibus):		0.000	Jarque-Bera (JB):		1743.73
5					
Skew:		0.123	Prob(JB):		0.0
0					
Kurtosis:		5.861	Cond. No.		5.85e+0
7					
<hr/>					
=					

#### Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 5.85e+07. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [72]: #statsmodels.stats.outliers_influence.variance - A measure for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_series1 = pd.Series([variance_inflation_factor(X_train1.values,i) for i in range(X_train1.shape[1])])
print('Train Model 1: \n\n{}\n'.format(vif_series1))
```

Train Model 1:

const	843215.673
Year	2.198
Kilometers_Driven	1.688
Mileage	2.122
Engine	6.520
Power	7.102
Seats	1.818
Transmission_Manual	2.120
Owner_Type_Fourth & Above	1.013
Owner_Type_Second	1.139
Owner_Type_Third	1.077
Region_North	2.482
Region_South	3.654
Region_West	3.109
Car_Type_Level2	1.669
Car_Type_Level3	2.284
Car_Type_Level4	3.052
Car_Type_Level5	1.641
Car_Type_Level6	1.116
dtype: float64	

```
In [73]: #training second model to display values
X_train2 = X_train1.drop('Power', axis=1)
vif_series2 = pd.Series([variance_inflation_factor(X_train2.values,i) for i in range(X_train2.shape[1])])
print('Train Model 2: \n\n{} \n'.format(vif_series2))
```

Train Model 2:

```
const          841255.110
Year           2.194
Kilometers_Driven      1.688
Mileage         2.093
Engine          3.925
Seats            1.625
Transmission_Manual    1.957
Owner_Type_Fourth & Above 1.012
Owner_Type_Second       1.138
Owner_Type_Third        1.077
Region_North          2.480
Region_South           3.638
Region_West            3.106
Car_Type_Level2        1.586
Car_Type_Level3        2.077
Car_Type_Level4        2.577
Car_Type_Level5        1.532
Car_Type_Level6        1.107
dtype: float64
```

## Get Regression Diagnostics Summary

In [74]: #OLS model - depicting values

```
olsmod1 = sm.OLS(y_train, X_train2)
olsres1 = olsmod1.fit()
print(olsres1.summary())
```

### OLS Regression Results

Dep. Variable:	Price	R-squared:	0.950			
Model:	OLS	Adj. R-squared:	0.949			
Method:	Least Squares	F-statistic:	5601.			
Date:	Sat, 11 Jun 2022	Prob (F-statistic):	0.00			
Time:	18:55:52	Log-Likelihood:	-8362.3			
No. Observations:	5075	AIC:	1.676e+04			
Df Residuals:	5057	BIC:	1.688e+04			
Df Model:	17					
Covariance Type:	nonrobust					
25	0.975]	coef	std err	t	P> t	[0.0
const		-392.6371	16.214	-24.216	0.000	-424.4
23	-360.851					
Year		0.1961	0.008	24.274	0.000	0.1
80	0.212					
Kilometers_Driven		-1.718e-06	7.61e-07	-2.258	0.024	-3.21e-
06	-2.27e-07					
Mileage		0.0320	0.006	5.160	0.000	0.0
20	0.044					
Engine		0.0016	6.19e-05	25.561	0.000	0.0
01	0.002					
Seats		-0.0793	0.028	-2.844	0.004	-0.1
34	-0.025					
Transmission_Manual		-0.9898	0.055	-18.024	0.000	-1.0
97	-0.882					
Owner_Type_Fourth & Above		0.4341	0.423	1.027	0.304	-0.3
95	1.263					
Owner_Type_Second		-0.0778	0.052	-1.488	0.137	-0.1
80	0.025					
Owner_Type_Third		-0.2048	0.135	-1.513	0.130	-0.4
70	0.061					
Region_North		0.1776	0.076	2.340	0.019	0.0
29	0.326					
Region_South		0.3221	0.067	4.772	0.000	0.1
90	0.454					
Region_West		0.2395	0.070	3.415	0.001	0.1
02	0.377					
Car_Type_Level2		2.7833	0.050	55.226	0.000	2.6
85	2.882					
Car_Type_Level3		8.9825	0.078	115.632	0.000	8.8
30	9.135					
Car_Type_Level4		12.7053	0.098	130.038	0.000	12.5
14	12.897					
Car_Type_Level5		11.6370	0.178	65.427	0.000	11.2
88	11.986					
Car_Type_Level6		11.1725	0.419	26.639	0.000	10.3

```
50      11.995
=====
Omnibus:          299.978   Durbin-Watson:        2.007
Prob(Omnibus):    0.000     Jarque-Bera (JB):  1076.358
Skew:              0.199     Prob(JB):           1.87e-234
Kurtosis:         5.221     Cond. No.          5.85e+07
=====
```

**Notes:**

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 5.85e+07. This might indicate that there are strong multicollinearity or other numerical problems.

In [75]: #Running third model - depicting values

```
X_train3 = X_train1.drop('Engine', axis=1)
vif_series3 = pd.Series([variance_inflation_factor(X_train3.values,i) for i in range(X_train3.shape[1])])
print('Train Model 3: \n\n{} \n'.format(vif_series3))
```

Train Model 3:

const	841834.515
Year	2.195
Kilometers_Driven	1.625
Mileage	2.013
Power	4.275
Seats	1.401
Transmission_Manual	2.114
Owner_Type_Fourth & Above	1.012
Owner_Type_Second	1.138
Owner_Type_Third	1.077
Region_North	2.478
Region_South	3.647
Region_West	3.102
Car_Type_Level2	1.647
Car_Type_Level3	2.235
Car_Type_Level4	3.010
Car_Type_Level5	1.590
Car_Type_Level6	1.096
dtype: float64	

In [76]: #Running OLS

```
olsmod2 = sm.OLS(y_train, X_train3)
olsres2 = olsmod2.fit()
print(olsres2.summary())
```

### OLS Regression Results

Dep. Variable:	Price	R-squared:	0.951			
Model:	OLS	Adj. R-squared:	0.951			
Method:	Least Squares	F-statistic:	5775.			
Date:	Sat, 11 Jun 2022	Prob (F-statistic):	0.00			
Time:	18:55:52	Log-Likelihood:	-8288.4			
No. Observations:	5075	AIC:	1.661e+04			
Df Residuals:	5057	BIC:	1.673e+04			
Df Model:	17					
Covariance Type:	nonrobust					
25	0.975]	coef	std err	t	P> t	[0.0
const		-398.3288	15.985	-24.919	0.000	-429.6
66	-366.992					
Year		0.1982	0.008	24.883	0.000	0.1
83	0.214					
Kilometers_Driven		-1.577e-07	7.36e-07	-0.214	0.830	-1.6e-
06	1.28e-06					
Mileage		0.0293	0.006	4.898	0.000	0.0
18	0.041					
Power		0.0224	0.001	28.673	0.000	0.0
21	0.024					
Seats		0.2117	0.026	8.292	0.000	0.1
62	0.262					
Transmission_Manual		-0.7115	0.056	-12.650	0.000	-0.8
22	-0.601					
Owner_Type_Fourth & Above		0.5719	0.417	1.373	0.170	-0.2
45	1.389					
Owner_Type_Second		-0.1023	0.052	-1.984	0.047	-0.2
03	-0.001					
Owner_Type_Third		-0.2410	0.133	-1.807	0.071	-0.5
03	0.021					
Region_North		0.1822	0.075	2.437	0.015	0.0
36	0.329					
Region_South		0.3623	0.067	5.440	0.000	0.2
32	0.493					
Region_West		0.2376	0.069	3.439	0.001	0.1
02	0.373					
Car_Type_Level2		2.6626	0.051	52.608	0.000	2.5
63	2.762					
Car_Type_Level3		8.7180	0.079	109.775	0.000	8.5
62	8.874					
Car_Type_Level4		12.1741	0.104	116.988	0.000	11.9
70	12.378					
Car_Type_Level5		11.2187	0.179	62.824	0.000	10.8
69	11.569					
Car_Type_Level6		11.1487	0.411	27.111	0.000	10.3

42 11.955

Omnibus:	363.497	Durbin-Watson:	1.998
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1637.494
Skew:	0.189	Prob(JB):	0.00
Kurtosis:	5.757	Cond. No.	5.85e+07

## Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 5.85e+07. This might indicate that there are strong multicollinearity or other numerical problems.

In [77]: #Running OLS

```
X_train4 = X_train3.drop('Kilometers_Driven', axis=1)
olsmod3 = sm.OLS(y_train, X_train4)
olsres3 = olsmod3.fit()
print(olsres3.summary())
```

### OLS Regression Results

Dep. Variable:	Price	R-squared:	0.951		
Model:	OLS	Adj. R-squared:	0.951		
Method:	Least Squares	F-statistic:	6137.		
Date:	Sat, 11 Jun 2022	Prob (F-statistic):	0.00		
Time:	18:55:52	Log-Likelihood:	-8288.4		
No. Observations:	5075	AIC:	1.661e+04		
Df Residuals:	5058	BIC:	1.672e+04		
Df Model:	16				
Covariance Type:	nonrobust				
	coef	std err	t	P> t	[0.0]
25	0.975]				
const	-399.9048	14.192	-28.179	0.000	-427.7
27	-372.083				
Year	0.1989	0.007	28.150	0.000	0.1
85	0.213				
Mileage	0.0292	0.006	4.917	0.000	0.0
18	0.041				
Power	0.0224	0.001	28.968	0.000	0.0
21	0.024				
Seats	0.2103	0.025	8.520	0.000	0.1
62	0.259				
Transmission_Manual	-0.7123	0.056	-12.697	0.000	-0.8
22	-0.602				
Owner_Type_Fourth & Above	0.5712	0.417	1.371	0.170	-0.2
46	1.388				
Owner_Type_Second	-0.1027	0.052	-1.993	0.046	-0.2
04	-0.002				
Owner_Type_Third	-0.2410	0.133	-1.807	0.071	-0.5
03	0.021				
Region_North	0.1794	0.074	2.437	0.015	0.0
35	0.324				
Region_South	0.3593	0.065	5.519	0.000	0.2
32	0.487				
Region_West	0.2356	0.068	3.442	0.001	0.1
01	0.370				
Car_Type_Level2	2.6630	0.051	52.663	0.000	2.5
64	2.762				
Car_Type_Level3	8.7183	0.079	109.813	0.000	8.5
63	8.874				
Car_Type_Level4	12.1763	0.104	117.588	0.000	11.9
73	12.379				
Car_Type_Level5	11.2217	0.178	63.042	0.000	10.8
73	11.571				
Car_Type_Level6	11.1522	0.411	27.144	0.000	10.3
47	11.958				

Omnibus:	363.483	Durbin-Watson:	1.998
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1637.000
Skew:	0.189	Prob(JB):	0.00
Kurtosis:	5.757	Cond. No.	1.64e+06

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.64e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [78]: `residual = olsres3.resid  
np.mean(residual)`

Out[78]: `-1.3657719946781153e-12`

In [79]: `residual=olsres3.resid  
fitted=olsres3.fittedvalues #predicted values`

Regression Results: Adjusted R-squared: Model fit between the range of 0 to 1 A high Adjusted R-Squared value of 0.953 indicates good fit.

Const coefficient is the Y-intercept with -40.48 Std err: coefficients reflect high accuracy P >|t|: It is p-value.

This shows that for each independent feature there is a null hypothesis and alternate hypothesis

H<sub>0</sub> : Independent variable is not significant

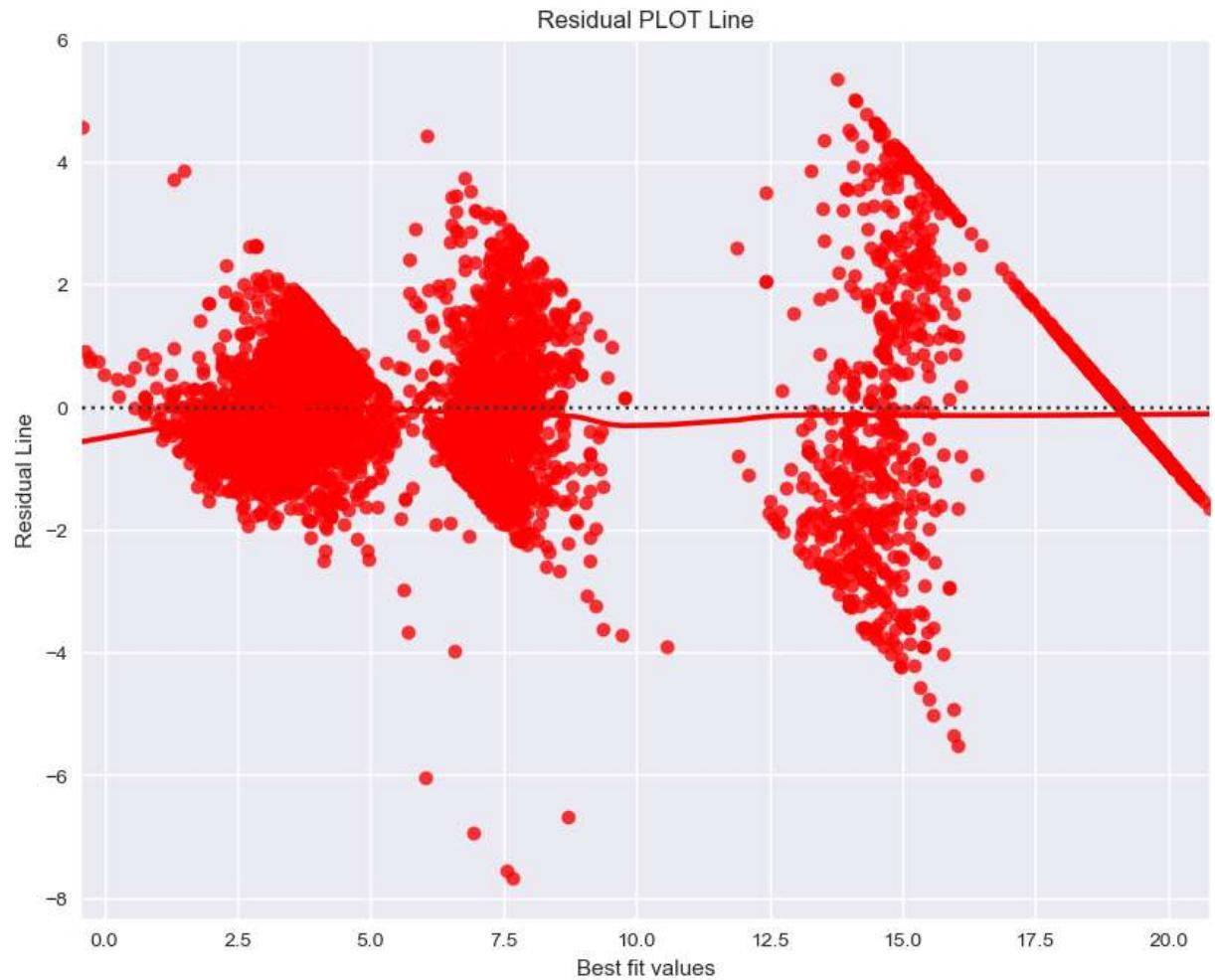
H<sub>a</sub> : Independent variable is significant

If p-value is less than 0.05 , then the variable is considered to be statistically significant.

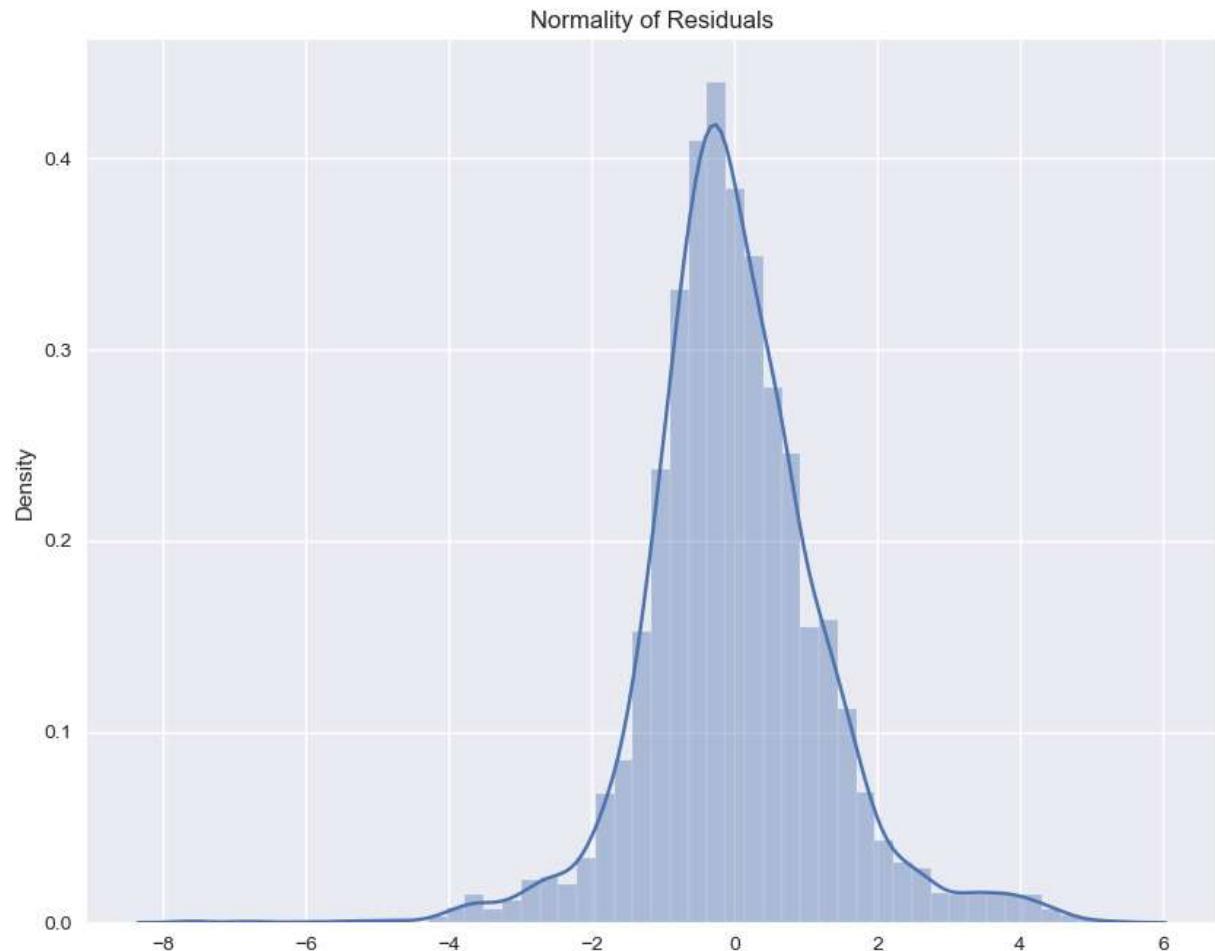
Confidence Interval 95%: It represents the range in which our coefficients are likely to fall.

## Draw a prediction line with data points on a scatter plot Showing Residuals Best Fit

```
In [80]: #GRaph mapping residual best plot / Line of fit
sns.set_style("darkgrid")
sns.residplot(fitted,residual,color="red",lowess=True)
plt.xlabel("Best fit values")
plt.ylabel("Residual Line")
plt.title("Residual PLOT Line")
plt.show()
```



```
In [81]: #Histogram of Residuals  
sns.distplot(residual)  
plt.title('Normality of Residuals')  
plt.show()
```

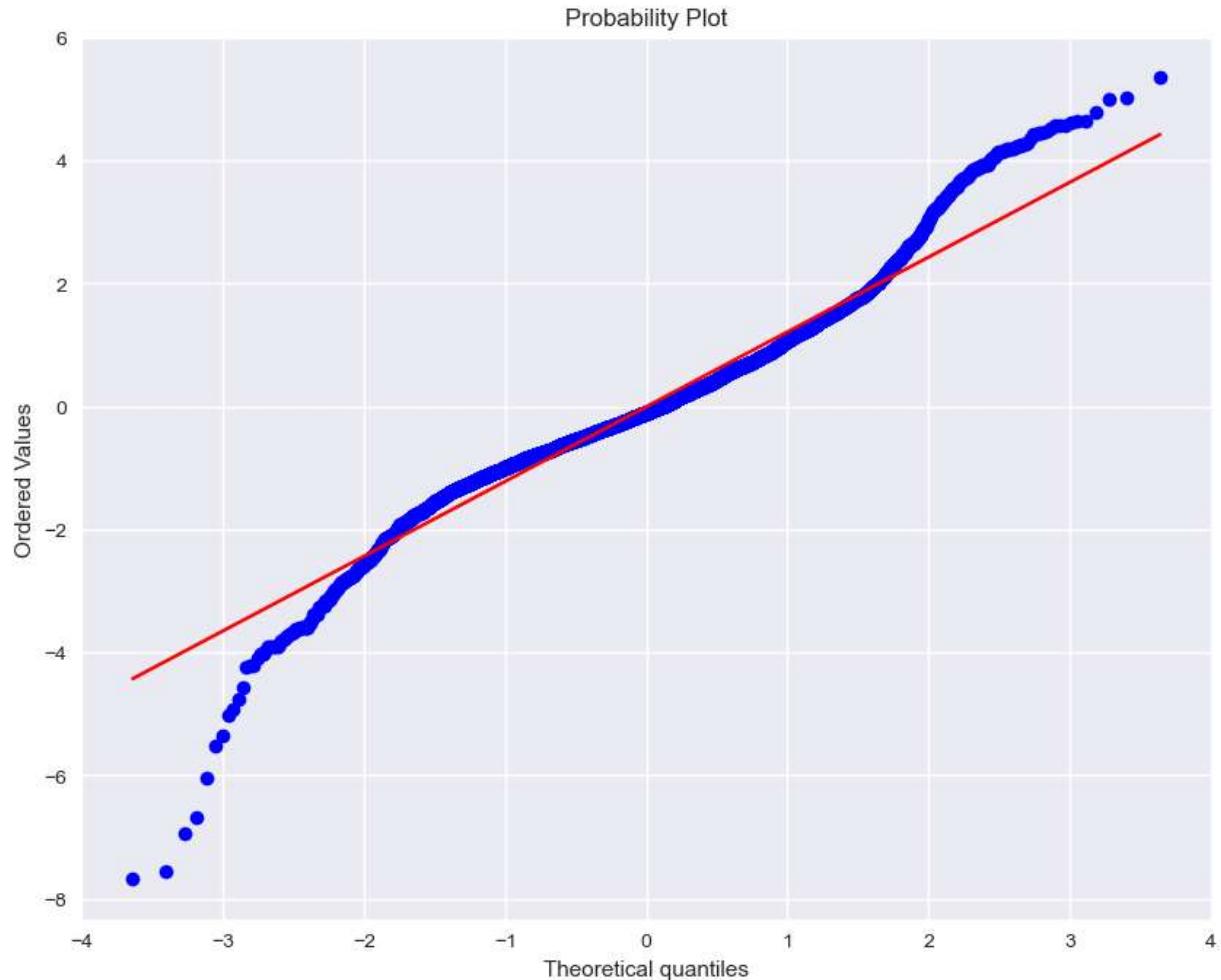


## Check the normality assumptions by creating a QQ-plot

QQ Plot shows linearity see below

In [82]: # Q-Q plot to check the normal probability of residuals.

```
import pylab
import scipy.stats as stats
stats.probplot(residual,dist="norm",plot=pylab)
plt.show()
```



In [83]: #Obtain F statistic and P values

```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(residual, X_train4)
lzip(name, test)
```

Out[83]: [('F statistic', 1.033807218416428), ('p-value', 0.20199670346246976)]

## Model Data for Prediction

In [84]: `#Clarify x train column detail`  
`X_train4.columns`

Out[84]: Index(['const', 'Year', 'Mileage', 'Power', 'Seats', 'Transmission\_Manual', 'Owner\_Type\_Fourth & Above', 'Owner\_Type\_Second', 'Owner\_Type\_Third', 'Region\_North', 'Region\_South', 'Region\_West', 'Car\_Type\_Level2', 'Car\_Type\_Level3', 'Car\_Type\_Level4', 'Car\_Type\_Level5', 'Car\_Type\_Level6'],  
`dtype='object')`

In [85]: `#Print x train head`  
`X_test_final = X_test[X_train4.columns]`  
`X_test_final.head()`

Out[85]:

	const	Year	Mileage	Power	Seats	Transmission_Manual	Owner_Type_Fourth & Above	Owner_Type
1083	1.000	2015	21.430	87.200	5.000		1	0
4601	1.000	2019	20.680	190.000	5.000		0	0
1969	1.000	2011	12.400	132.000	8.000		1	0
5344	1.000	2015	18.600	81.830	5.000		1	0
2056	1.000	2015	16.770	229.575	5.000		0	0

In [86]: `#set variable predictor value`  
`y_pred = olsres3.predict(X_test_final)`

## Cross Validation Using Sci-kit Learn

In [87]: `#Checking root mean squared error for train and test set`

```
rms = np.sqrt(mean_squared_error(y_train, fitted))
print('Train error:',rms)

rms1 = np.sqrt(mean_squared_error(y_test, y_pred))
print('Test error:',rms1)
```

Train error: 1.2389217848929732  
Test error: 1.2327474340985183

Train and test values are very low, yet is not over / under fitting.

In [88]: #final train test

```
olsmodtest = sm.OLS(y_test, X_test_final)
olsrestest = olsmodtest.fit()
print(olsrestest.summary())
```

### OLS Regression Results

Dep. Variable:	Price	R-squared:	0.953			
Model:	OLS	Adj. R-squared:	0.952			
Method:	Least Squares	F-statistic:	2720.			
Date:	Sat, 11 Jun 2022	Prob (F-statistic):	0.00			
Time:	18:55:56	Log-Likelihood:	-3532.1			
No. Observations:	2176	AIC:	7098.			
Df Residuals:	2159	BIC:	7195.			
Df Model:	16					
Covariance Type:	nonrobust					
25	0.975]	coef	std err	t	P> t	[0.0
const		-415.2947	21.617	-19.212	0.000	-457.6
87	-372.903					
Year		0.2061	0.011	19.132	0.000	0.1
85	0.227					
Mileage		0.0388	0.009	4.209	0.000	0.0
21	0.057					
Power		0.0237	0.001	20.555	0.000	0.0
21	0.026					
Seats		0.3157	0.038	8.271	0.000	0.2
41	0.391					
Transmission_Manual		-0.7067	0.084	-8.402	0.000	-0.8
72	-0.542					
Owner_Type_Fourth & Above		-0.0296	0.716	-0.041	0.967	-1.4
35	1.375					
Owner_Type_Second		-0.1154	0.076	-1.526	0.127	-0.2
64	0.033					
Owner_Type_Third		-0.4693	0.201	-2.338	0.019	-0.8
63	-0.076					
Region_North		0.2735	0.111	2.468	0.014	0.0
56	0.491					
Region_South		0.5520	0.098	5.605	0.000	0.3
59	0.745					
Region_West		0.2863	0.102	2.803	0.005	0.0
86	0.487					
Car_Type_Level2		2.6065	0.077	33.879	0.000	2.4
56	2.757					
Car_Type_Level3		8.6726	0.117	74.426	0.000	8.4
44	8.901					
Car_Type_Level4		12.0980	0.154	78.776	0.000	11.7
97	12.399					
Car_Type_Level5		11.2200	0.247	45.392	0.000	10.7
35	11.705					
Car_Type_Level6		11.4906	0.890	12.905	0.000	9.7
44	13.237					

Omnibus:	197.434	Durbin-Watson:	1.967
Prob(Omnibus):	0.000	Jarque-Bera (JB):	495.288
Skew:	0.521	Prob(JB):	2.82e-108
Kurtosis:	5.092	Cond. No.	1.65e+06

---

**Notes:**

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.65e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [89]: `#print trained OLS model  
print(olsres3.summary())`

OLS Regression Results						
		Dep. Variable:	Price	R-squared:	0.95	
1		Model:	OLS	Adj. R-squared:	0.95	
1		Method:	Least Squares	F-statistic:	613	
7.		Date:	Sat, 11 Jun 2022	Prob (F-statistic):	0.0	
0		Time:	18:55:56	Log-Likelihood:	-8288.	
4		No. Observations:	5075	AIC:	1.661e+0	
4		Df Residuals:	5058	BIC:	1.672e+0	
4		Df Model:	16	Covariance Type:	nonrobust	
		coef	std err	t	P> t	
[0.025	0.975]					
7.727	-372.083	-399.9048	14.192	-28.179	0.000	-42
0.185	0.213	0.1989	0.007	28.150	0.000	
0.018	0.041	0.0292	0.006	4.917	0.000	
0.021	0.024	0.0224	0.001	28.968	0.000	
0.162	0.259	0.2103	0.025	8.520	0.000	
0.822	-0.602	-0.7123	0.056	-12.697	0.000	-
0.246	1.388	0.5712	0.417	1.371	0.170	-
0.204	-0.002	-0.1027	0.052	-1.993	0.046	-
0.503	0.021	-0.2410	0.133	-1.807	0.071	-
0.035	0.324	0.1794	0.074	2.437	0.015	
0.232	0.487	0.3593	0.065	5.519	0.000	
0.101	0.370	0.2356	0.068	3.442	0.001	
2.564	2.762	2.6630	0.051	52.663	0.000	
8.563	8.874	8.7183	0.079	109.813	0.000	
Car_Type_Level4		12.1763	0.104	117.588	0.000	1

```

1.973      12.379
Car_Type_Level5          11.2217    0.178    63.042    0.000    1
0.873      11.571
Car_Type_Level6          11.1522    0.411    27.144    0.000    1
0.347      11.958
=====
=
Omnibus:                  363.483   Durbin-Watson:           1.99
8
Prob(Omnibus):            0.000    Jarque-Bera (JB):        1637.00
0
Skew:                      0.189    Prob(JB):                0.0
0
Kurtosis:                 5.757    Cond. No.               1.64e+0
6
=====
=

```

**Notes:**

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.64e+06. This might indicate that there are strong multicollinearity or other numerical problems.

```
('F statistic', 1.033807218416428)
```

```
('p-value', 0.20199670346246976)]
```

```
##Presentation
```

```
#editing powerpoint
##record pptx
##print the pdf pptx slides
```

**Conclusion:**

Main impacts to buying Used cars in Dubai

The Ordinary Least Squares is a good model for prediction and inference at 95.1% adjusted Rsquared value.

Transmission and Owner type depict negative correlation to Price.

Automatic transmissions tend to be more expensive and manuals sell for less according to this model.

The year, mileage, power and number of seats have shown a positive assosiation with sale price.

