



Used Cars Analysis - Capstone Project

Import Necessary Libraries

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')
```

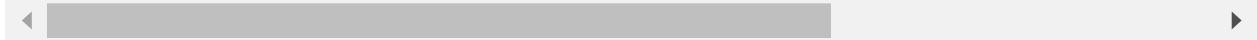
Data Understanding

```
In [2]: #Review the dataset  
data = pd.read_csv('data\\used_cars_data.csv')
```

```
In [3]: #Load the first rows  
data.head()
```

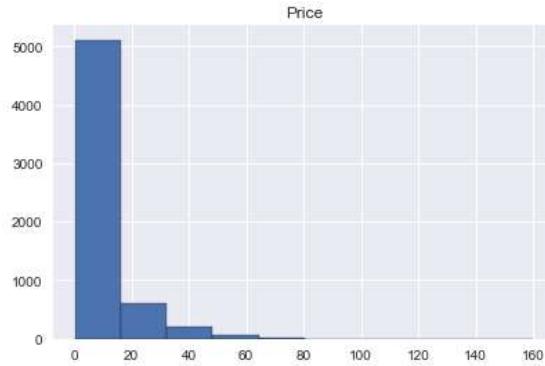
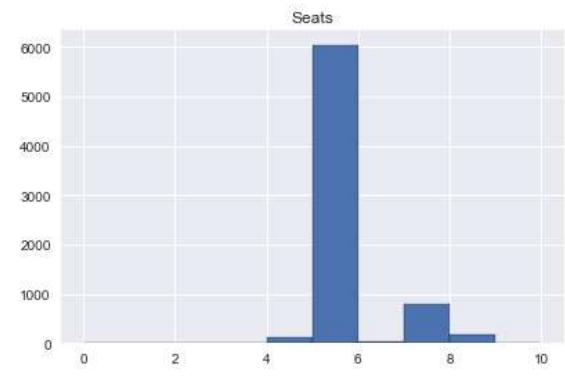
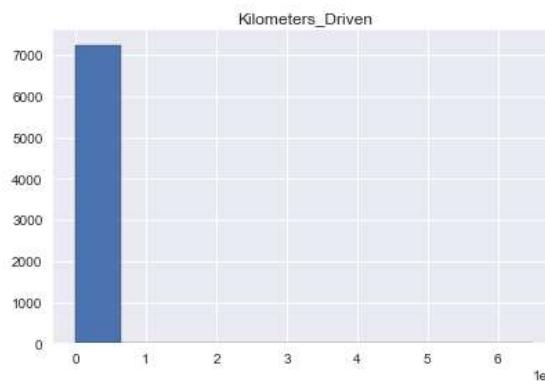
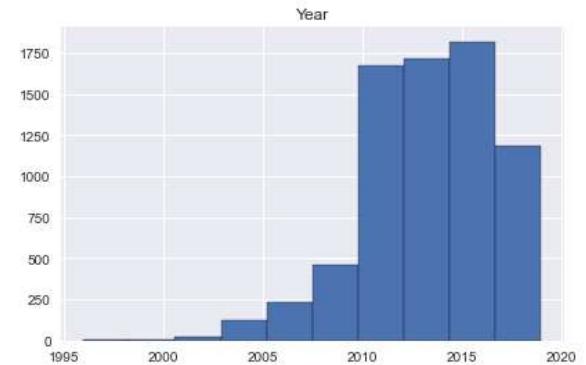
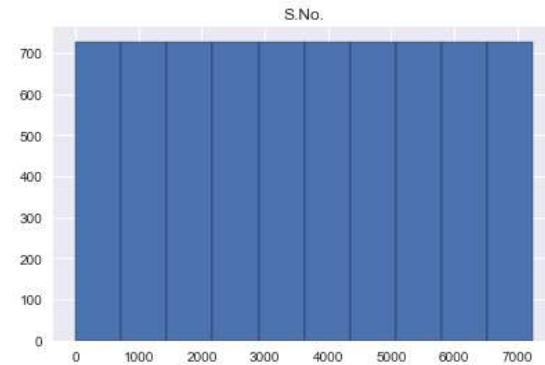
Out[3]:

	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 [4]: #Distribution visualization

```
data.hist(figsize=(15,15), edgecolor = 'black');
```



Data Preparation

```
In [5]: #Dropping the first column as its an ID place holder  
data.drop(['S.No.'],axis=1,inplace=True)
```

```
In [6]: #Drop the New Price column - no need for 2 Pricing structures  
data.drop(['New_Price'],axis=1, inplace=True)
```

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

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

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

```
In [9]: #Loop will add all the columns we want to change obj to string to use for a list  
num_values = []  
  
for colname in data.columns[data.dtypes == 'object']: # only need to focus on str  
    if data[colname].str.endswith('pl').any() or data[colname].str.endswith('kg')  
        # 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 the string from the numerical values in the column
##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:
    data[colname] = data[colname].apply(obj_to_num)#applying above function to the column
    data[colname]=data[colname].replace(0.0,np.nan)
```

Transform categorical columns for use in Linear Regression model

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

In [12]: #Checking dataset for non-null values, types, totals

```
data.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 category dataframe whilst viewing statistics

```
data.describe(include=["category"]).T
```

Out[13]:

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

OBSERVATIONS: -Mumbai sold the most cars -Preference of vehicle by 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

```
data[['Car_Brand','Model']] = data.Name.str.split(n=1,expand=True)
```

In [15]: #Model names are unique to the Car Brands

```
Brand_name=data['Car_Brand'].unique()
Model=data['Model'].unique()
```

```
In [16]: #Revising names
data['Car_Brand']=data['Car_Brand'].replace('Land','Land_Rover')
data['Car_Brand']=data['Car_Brand'].replace('ISUZU','Isuzu')
data['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 = data.select_dtypes(include=np.number).columns.tolist()
numeric_columns
```

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

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

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

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

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

```
In [22]: #Running a Loop to check every row in data dataset
for i in range(len(data)):
    if data.Price[i]==0.00:
        for j in range(len(Model)):
            if data.Model[i]==Model[j]: #Comparing the Car model names in both
                data.Price[i]=Median2[j] #replacing the Price of the car with th
```

```
In [23]: #Additional check for df types and confirming non-null values
data.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
data[data['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]: #Replacing the missing values with float 0.0
data['Price']= data['Price'].fillna(0.0)
#running a loop to check row in dataset
for i in range(len(data)):
    if data.Price[i]==0.00:
        for j in range(len(Brand_name)):
            if data.Car_Brand[i]==Brand_name[j]: #Comparing the brand names in both
                data.Price[i]=Median1[j] #replacing with corresponding median
```

```
In [26]: #Checking NaN Price values
data[data['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 cars in above df - (as they are rare, do not want to impact overall)
data.dropna(axis=0,inplace=True)
```

```
In [28]: #checking the shape
data.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)
data.describe().T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	95%
Year	7251.000	2013.369	3.247	1998.000	2011.000	2014.000	2016.000	2020.000
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	2500.000
Power	7251.000	112.318	52.929	34.200	77.000	94.000	138.030	200.000
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

Correlation Visualization

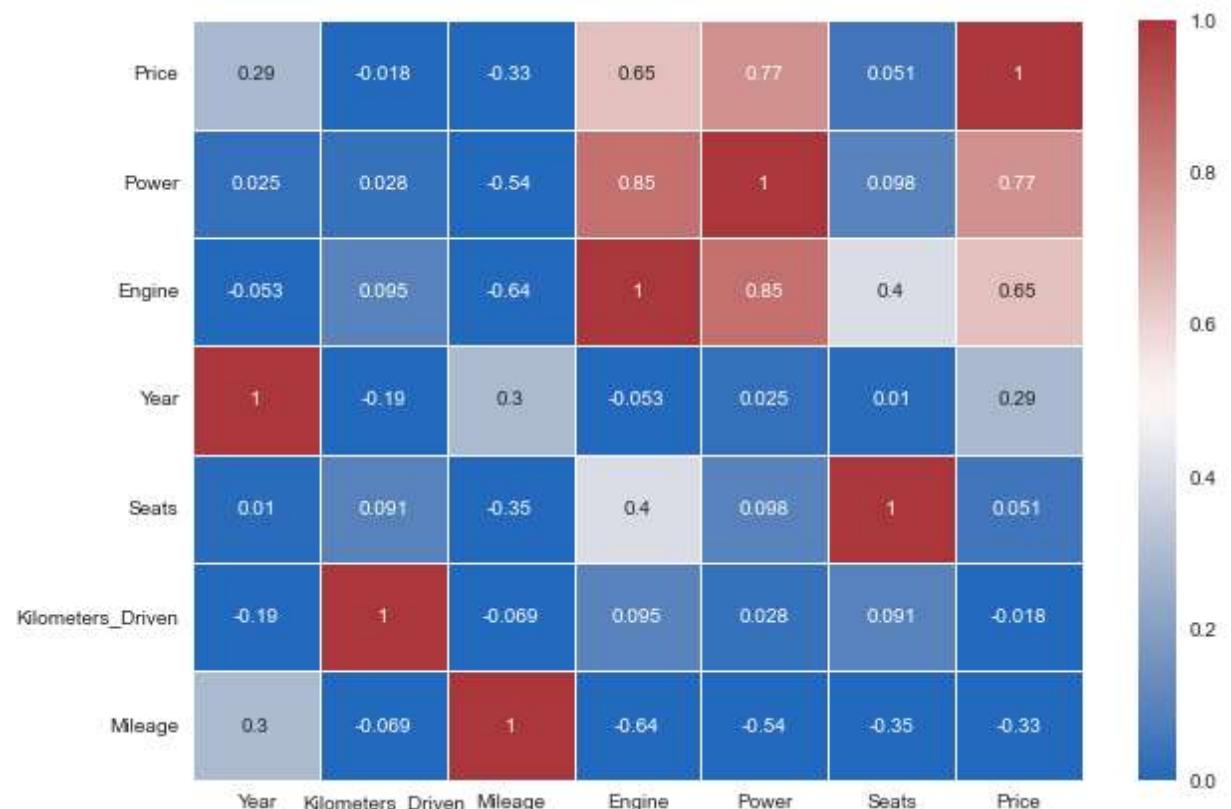
In [30]: `data.corr()`

Out[30]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Price
Year	1.000	-0.188	0.295	-0.053	0.025	0.010	0.292
Kilometers_Driven	-0.188	1.000	-0.069	0.095	0.028	0.091	-0.018
Mileage	0.295	-0.069	1.000	-0.638	-0.543	-0.348	-0.325
Engine	-0.053	0.095	-0.638	1.000	0.855	0.403	0.652
Power	0.025	0.028	-0.543	0.855	1.000	0.098	0.767
Seats	0.010	0.091	-0.348	0.403	0.098	1.000	0.051
Price	0.292	-0.018	-0.325	0.652	0.767	0.051	1.000

In [31]: `#heat map showing strongest correlations`

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

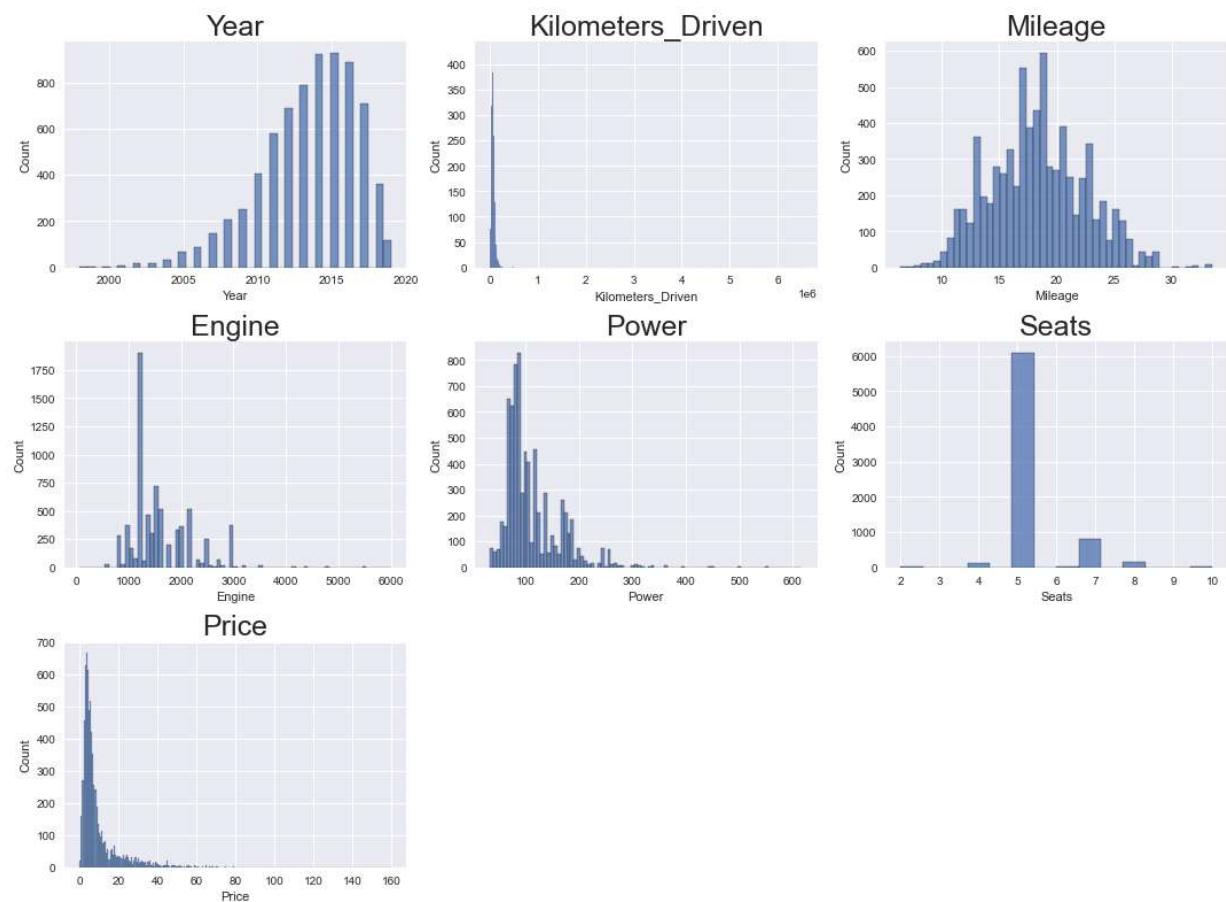


Univariate Analysis to Plot histograms to check the distributions of the predictors

In [32]: #Plotting histogram for distribution

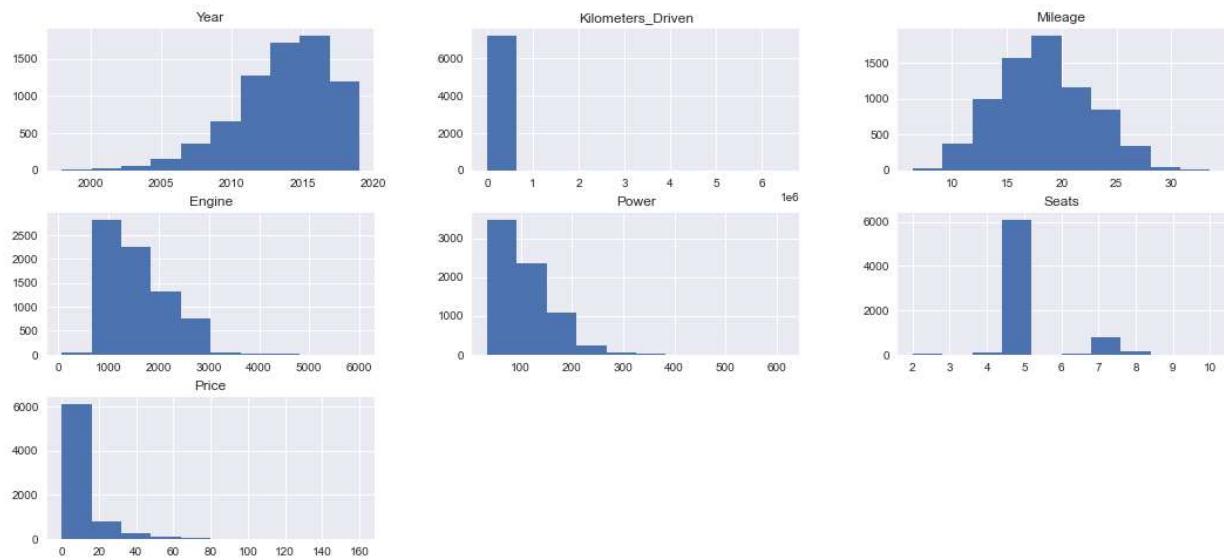
```
Uni_num = data.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(15,65))
#creating a loop that will show the plots for the columns in one plot
for i in range(len(Uni_num)):
    plt.subplot(18,3,i+1)
    sns.histplot(data[Uni_num[i]],kde=False)
    plt.tight_layout()
    plt.title(Uni_num[i],fontsize=25)

plt.show()
```



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

```
In [33]: #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()
data.hist(ax = ax);
```



Linear Regression Assumption checks

Linearity Normality (residuals) Homoscedasticity

```
In [34]: f = 'Price~Power'
```

```
In [35]: model = ols(formula=f, data=data).fit()
```

In [36]: model.summary()

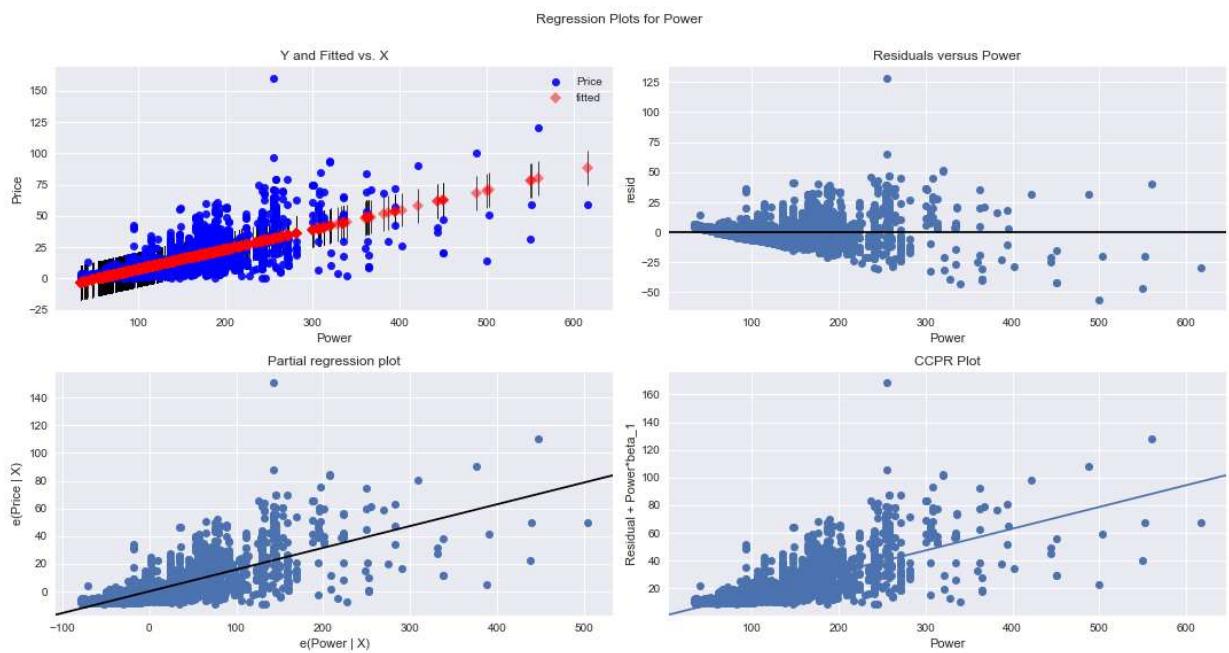
Out[36]: OLS Regression Results

Dep. Variable:	Price	R-squared:	0.588			
Model:	OLS	Adj. R-squared:	0.588			
Method:	Least Squares	F-statistic:	1.036e+04			
Date:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:35:59	Log-Likelihood:	-24367.			
No. Observations:	7251	AIC:	4.874e+04			
Df Residuals:	7249	BIC:	4.875e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	-8.3521	0.192	-43.488	0.000	-8.729	-7.976
Power	0.1574	0.002	101.766	0.000	0.154	0.160
Omnibus:	4151.352	Durbin-Watson:		2.026		
Prob(Omnibus):	0.000	Jarque-Bera (JB):		207758.991		
Skew:	2.036	Prob(JB):		0.00		
Kurtosis:	28.905	Cond. No.		291.		

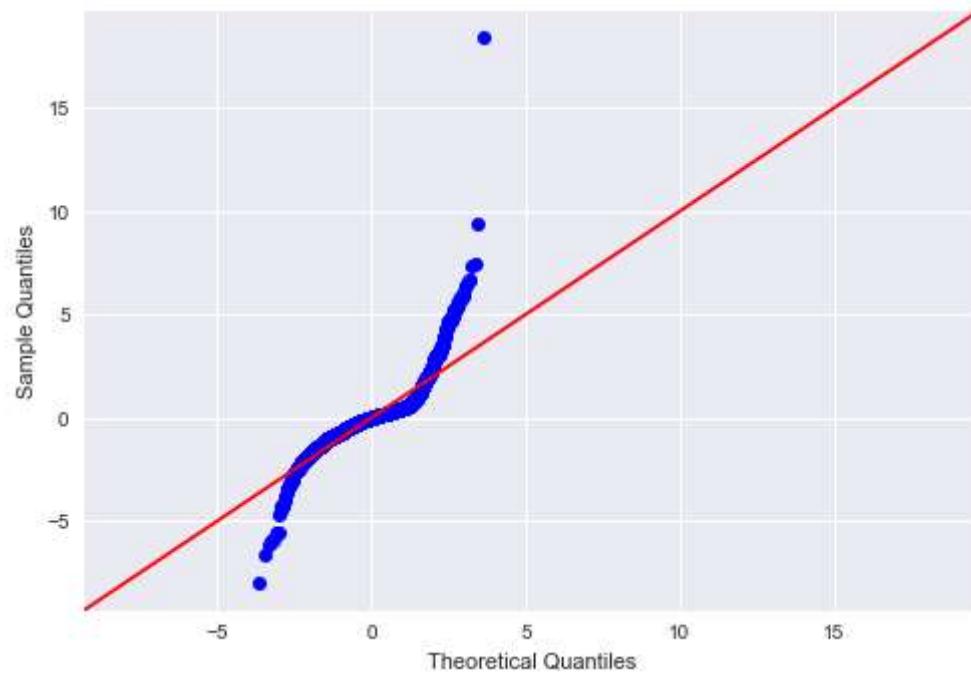
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [37]: fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "Power", fig=fig)
plt.show()
```



```
In [38]: import scipy.stats as stats
residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
fig.show()
```



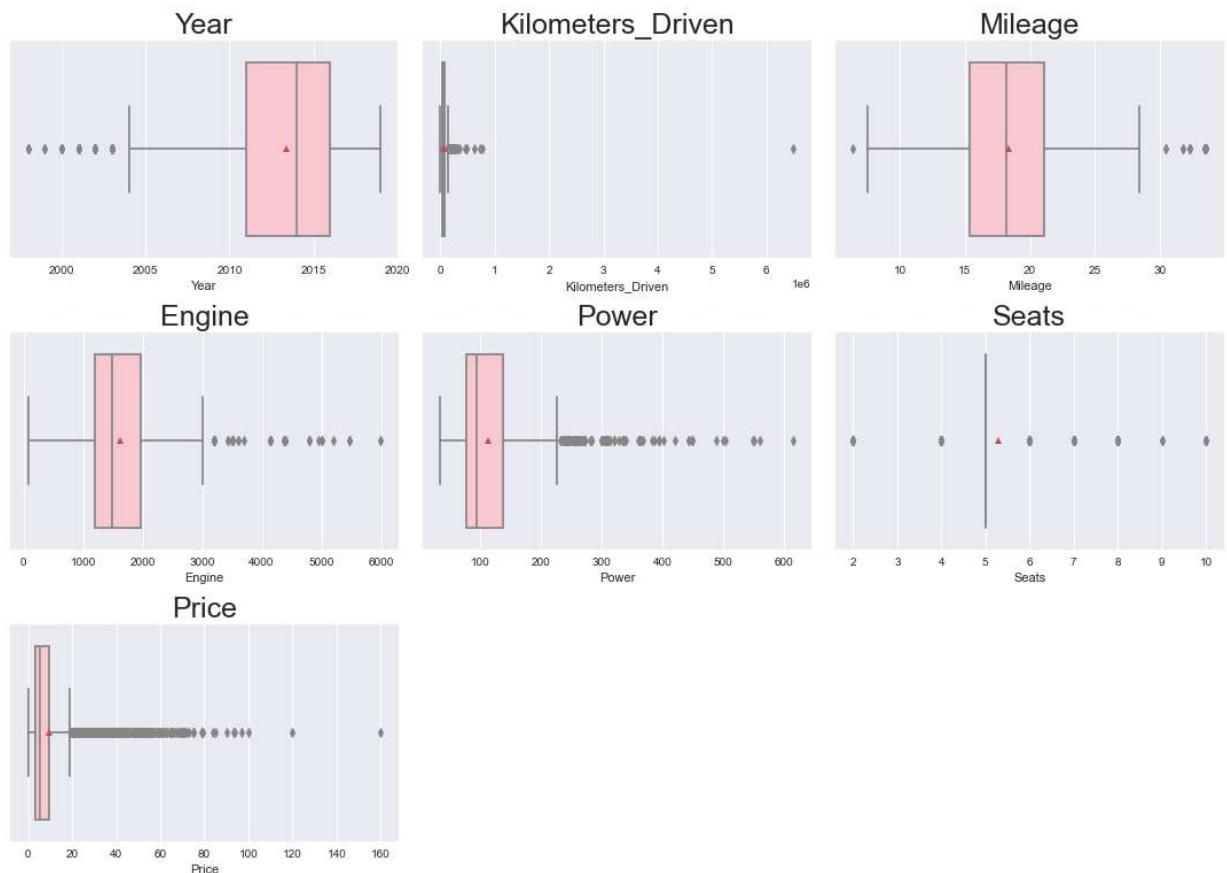
In [39]: #Shows

```
plt.scatter(data.Price, data.Power)
plt.title("Linearity check")
plt.show()
```



In [40]: *#Plotting a box plot to confirm assumptions*

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



More Data Preparation is required for categorical values

```
In [41]: #Creating region lists
regions ={'Delhi':'North','Jaipur':'North',
          'Chennai':'South','Coimbatore':'South','Hyderabad':'South','Bangalore':
          'Kolkata':'East',
          'Mumbai':'West','Pune':'West','Ahmedabad':'West'}
data['Region']=data['Location'].replace(regions)
```

```
In [42]: #Dropping unnecessary models & counting valid car type values to classify by Level
data.drop(["Car_Brand","Model"],axis=1,inplace=True)
data['Car_Type'] = pd.cut(data['Price'],[-np.inf,5.5,10.5,20.5,45.0,75.0,np.inf],
                           labels=["Level1","Level2","Level3","Level4","Level5","Level6"])
data['Car_Type'].value_counts()

## LEVELS = CAR_TYPE Level 1 is Low grade(economical Less expensive car) to Level 6 is High Grade
## Owner Type = First second cars, second card, third car, etc.
```

```
Out[42]: Level1    3638
Level2    1916
Level3     896
Level4     669
Level5     120
Level6      12
Name: Car_Type, dtype: int64
```

```
In [43]: #View sample of Level 3 type vehicle below
data.sample()
```

Out[43]:

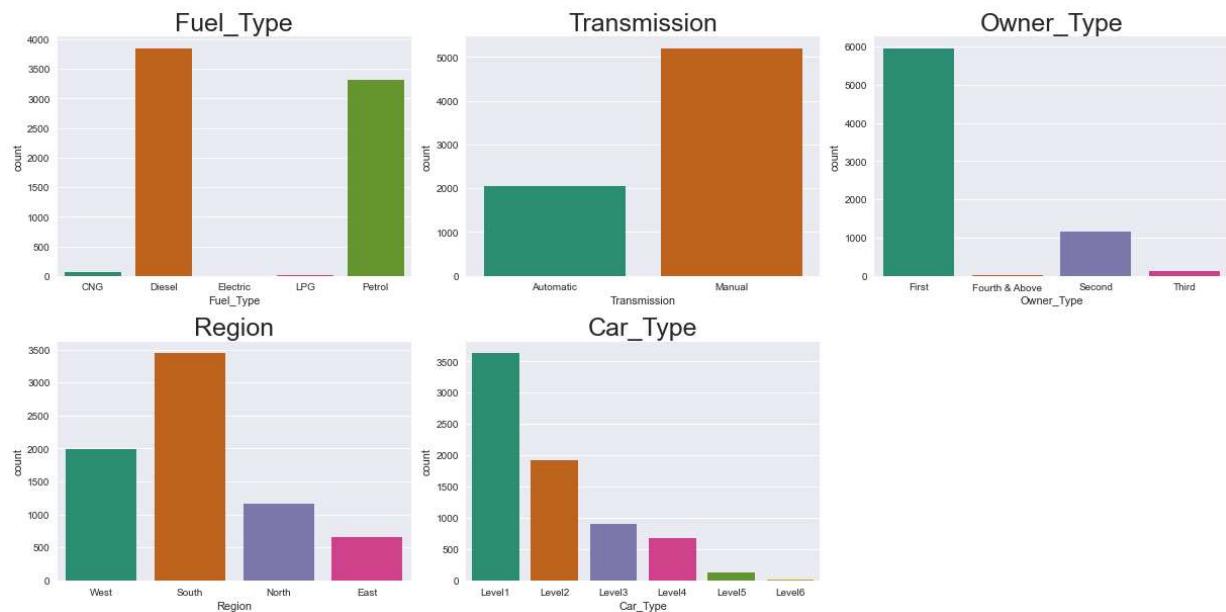
		Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
	3945	Skoda Octavia Elegance 2.0 TDI AT	Bangalore	2013	50000	Diesel	Automatic	First	19.300



```
In [44]: #Set categorical values to populate column list
categorical_val = data.select_dtypes(exclude=np.number).columns.tolist()
categorical_val.remove('Name')
categorical_val.remove('Location')
```

In [45]:

```
#visual bar graph
plt.figure(figsize=(17,75))
for i in range(len(categorical_val)):
    plt.subplot(18,3,i+1)
    ax=sns.countplot(data[categorical_val[i]],palette='Dark2')
    plt.tight_layout()
    plt.title(categorical_val[i],fontsize=25)
    total = len (data[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 [46]:

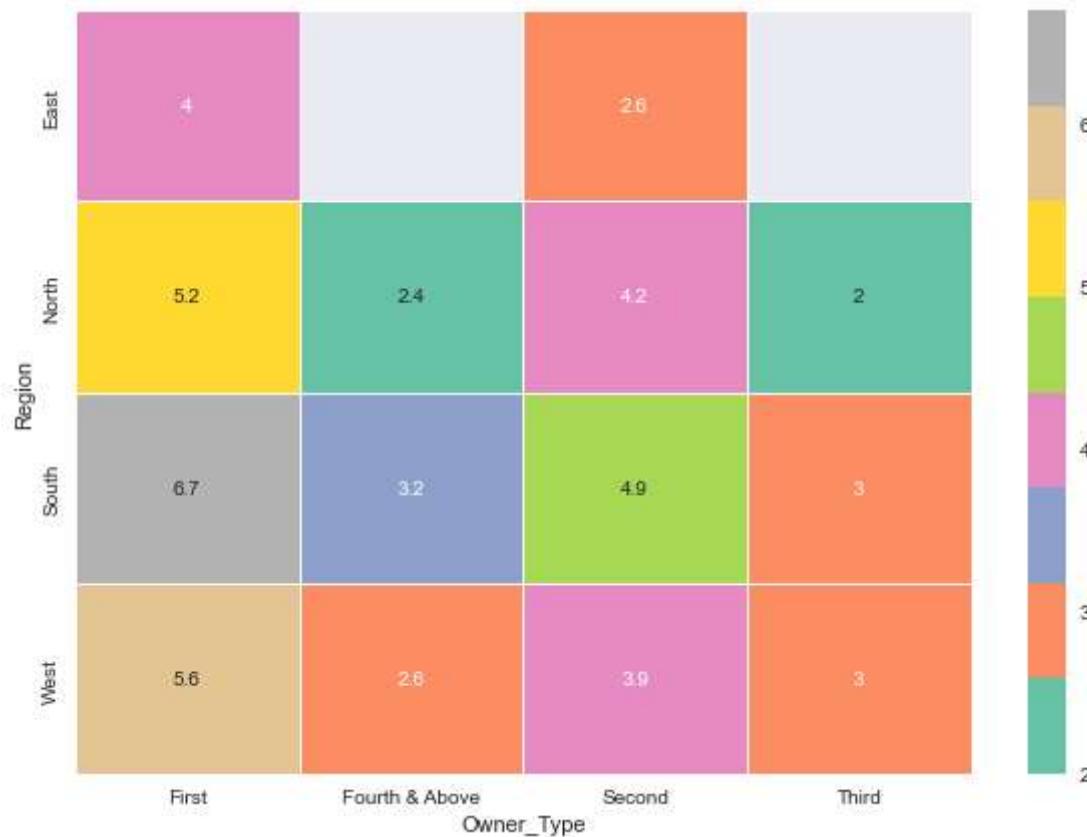
```
#Reviewing dataset for electric cars
data[data['Fuel_Type']=='Electric']
```

Out[46]:

		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 [47]: *#Does type of ownership affect Car price?*

```
df_hm = data.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
```



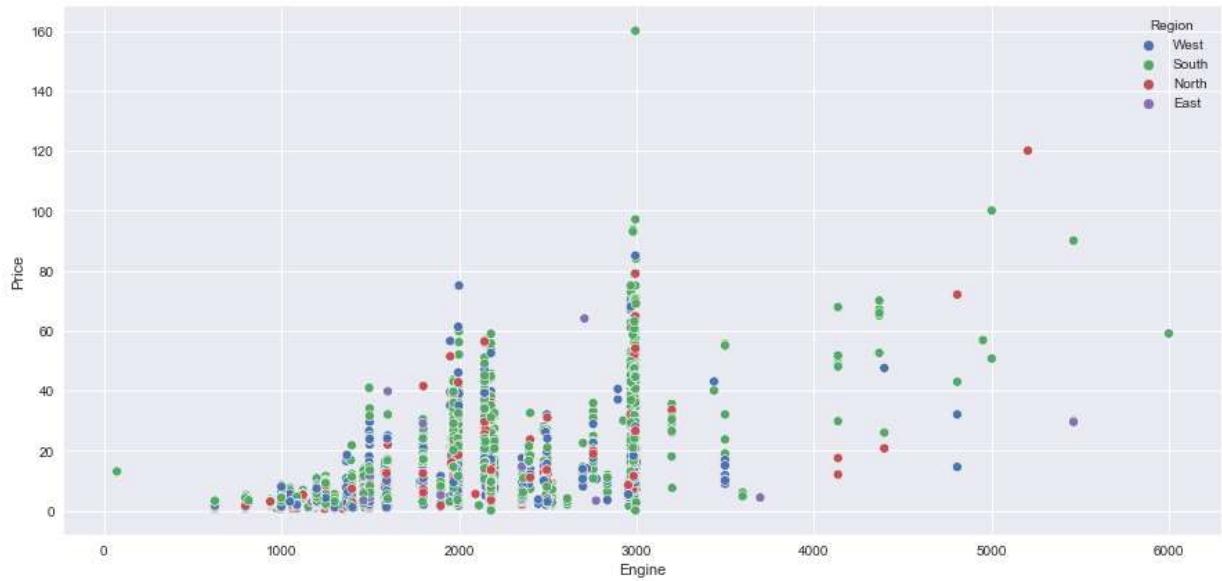
Linear Regression Assumption Checks with Visualisations

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

1. Linearity
2. Normality (residuals)
3. Homoscedasticity

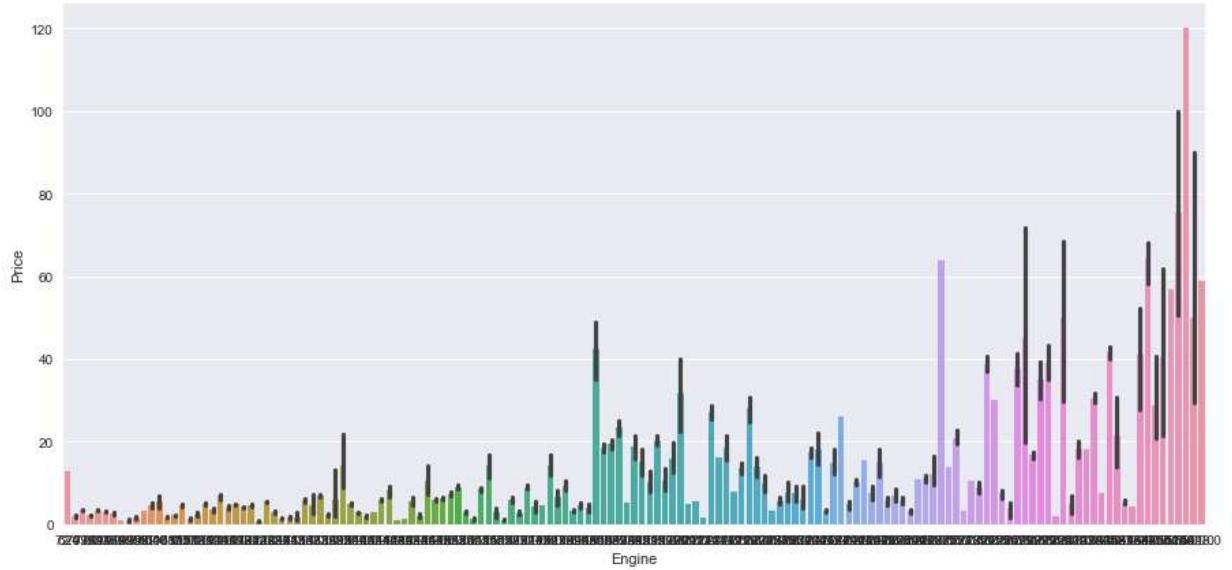
additional detail further incorporating test data

```
In [48]: #Analysis of variables that have high correlation with Price
#Price Vs Engine Vs Region
plt.figure(figsize=(15,7))
sns.scatterplot(data=data,y='Price',x='Engine',hue='Region')
plt.show()
```



```
In [49]: #Does type of Fuel affect car price?
plt.figure(figsize=(15,7))
sns.barplot(data=data,x='Engine',y='Price')
```

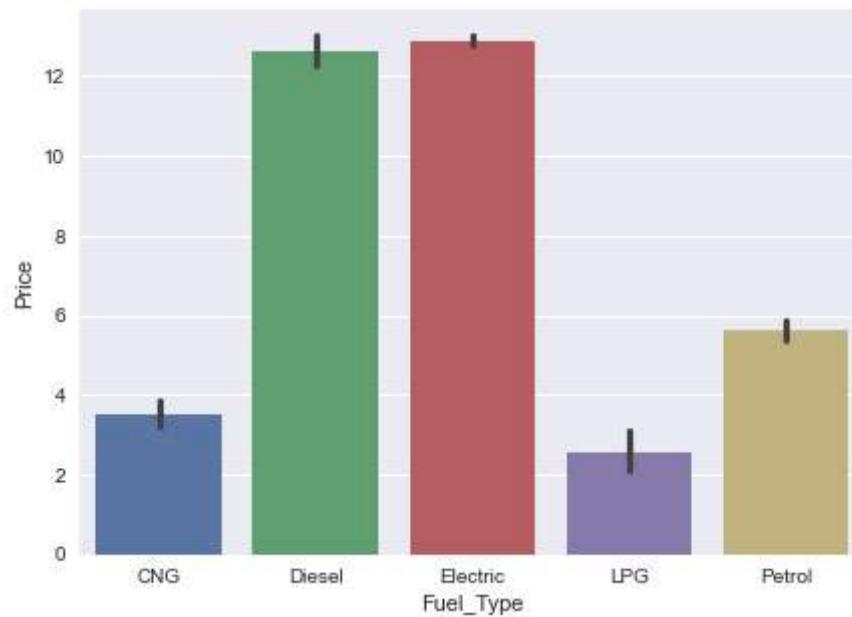
Out[49]: <AxesSubplot:xlabel='Engine', ylabel='Price'>



In [50]: #Does type of Fuel affect car price?

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

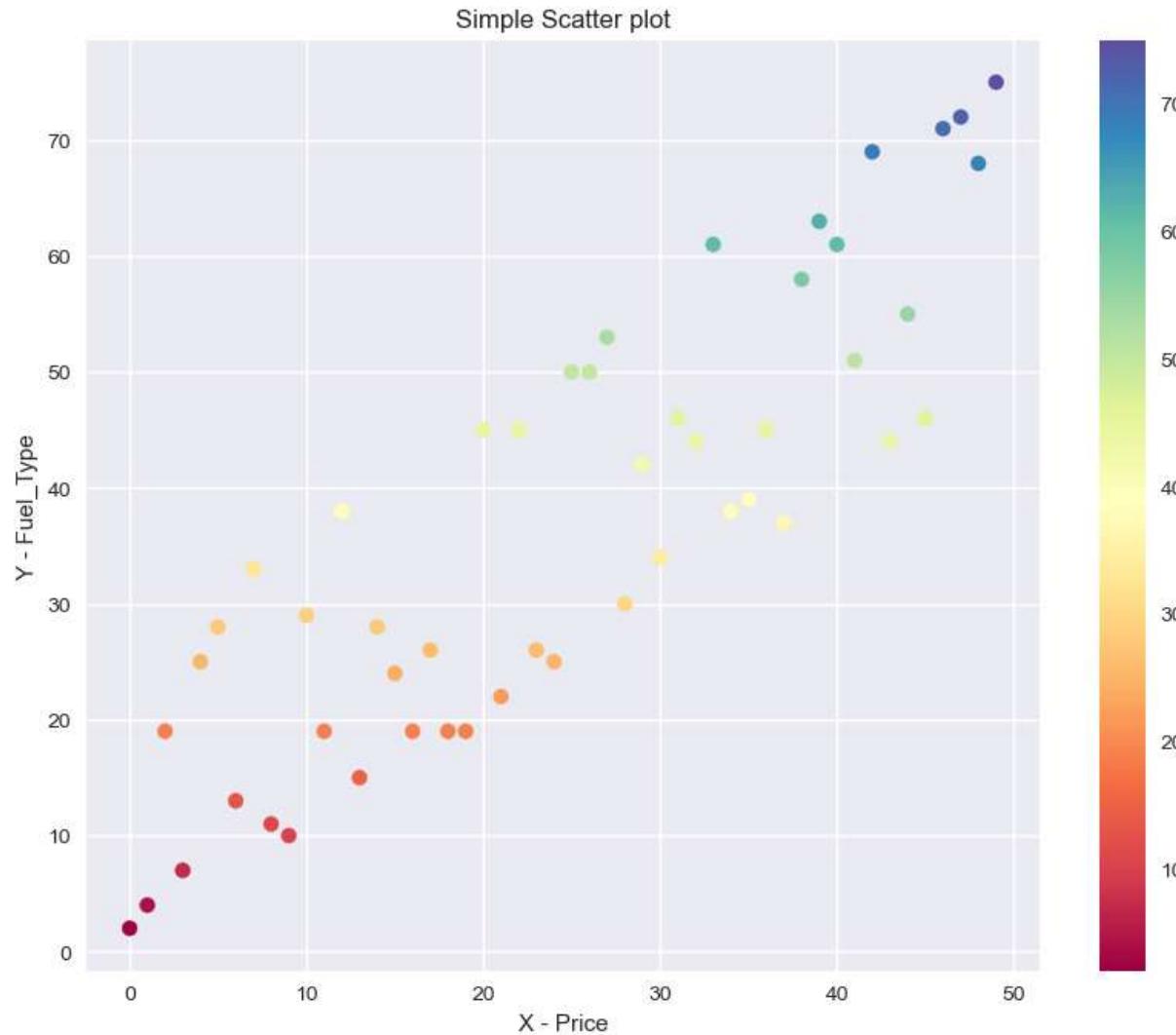
Out[50]: <AxesSubplot:xlabel='Fuel_Type', ylabel='Price'>



Visualize the error term for variance and heteroscedasticity

In [51]: # 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 [52]: #Dealing with outliers utilising the 25th & 75th quantiles
def treat_outliers(data,col):

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

    return data

def treat_outliers_all(data, col_list):

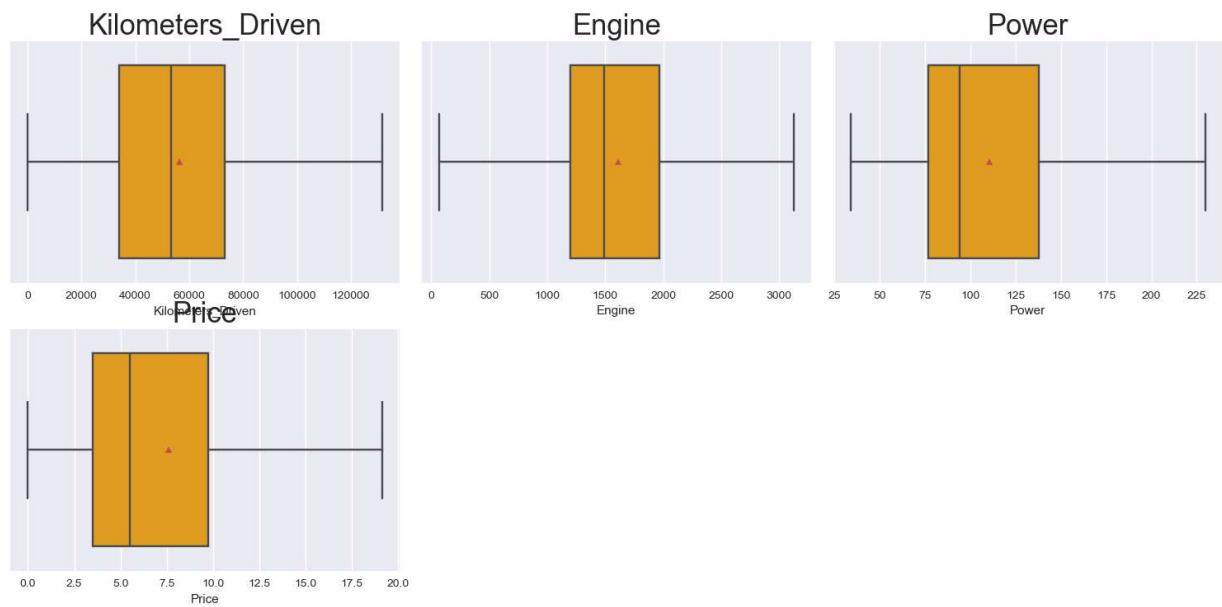
    for c in col_list:
        data = treat_outliers(data,c)
    return data
```

```
In [53]: #making a first copy & dropping Year, mileage and seats outliers as irrelevant
df2=data.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[53]: ['Kilometers_Driven', 'Engine', 'Power', 'Price']
```

```
In [54]: #dealing with outliers
df2 = treat_outliers_all(df2,numerical_col)
```

```
In [55]: #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 [56]: #Reviewing data head
df2.head()

Out[56]:

	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 [57]: #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 [58]: ##pd.set_option('display.float_format', Lambda x: '%.3f' % x) # to display number
df2.describe().T

Out[58]:

	count	mean	std	min	25%	50%	75%	
Year	7251.000	2013.369	3.247	1998.000	2011.000	2014.000	2016.000	2018.000
Kilometers_Driven	7251.000	56277.029	30191.137	171.000	34000.000	53416.000	73000.000	131500.000
Mileage	7251.000	18.346	4.134	6.400	15.400	18.200	21.100	24.000
Engine	7251.000	1608.204	563.734	72.000	1198.000	1493.000	1968.000	3100.000
Power	7251.000	110.408	45.953	34.200	77.000	94.000	138.030	200.000
Seats	7251.000	5.278	0.807	2.000	5.000	5.000	5.000	5.000
Price	7251.000	7.594	5.612	0.000	3.500	5.500	9.750	19.000

In [59]:

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

Out[59]:

	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 [60]:

```
#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_state=42)
X_train.head()
```

Out[60]:

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Transmission_Manual	Owner_Type
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 [61]: #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_))
```

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

```
In [63]: # Mean Absolute Error on test
mean_absolute_error(y_test, pred)
```

```
Out[63]: 0.8903015607952957
```

```
In [64]: # RMSE on test data
mean_squared_error(y_test, pred)**0.5
```

```
Out[64]: 1.215184412886623
```

```
In [65]: # R-squared on test rather than Log transformation as data is not highly skewed
r2_score(y_test, pred)
```

```
Out[65]: 0.9536193921069149
```

```
In [66]: #Calculating train score
linearregression.score(X_train, y_train) # 70 % data
```

```
Out[66]: 0.9520061630151408
```

In [67]: `#Calculating Test Score
linearregression.score(X_test, y_test) # unseen data`

Out[67]: 0.9536193921069149

In [68]: `#Creating another copy of df
df3=data.copy()
df3.head()`

Out[68]:

	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 [69]: `#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 [70]: `#Creating Dummy Variables for categorical columns
X1 = pd.get_dummies(X1, columns=['Region', 'Car_Type', 'Transmission', 'Owner_Type'])
X1.head()`

Out[70]:

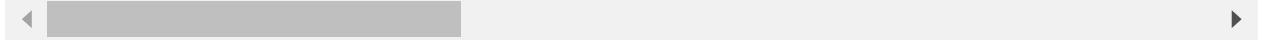
	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Region_North	Region_South	Region_East
0	2010	72000	26.600	998.000	58.160	5.000	0	0	0
1	2015	41000	19.670	1582.000	126.200	5.000	0	0	0
2	2011	46000	18.200	1199.000	88.700	5.000	0	1	0
3	2012	87000	20.770	1248.000	88.760	7.000	0	1	0
4	2013	40670	15.200	1968.000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

◀ ▶

```
In [71]: #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[71]:

	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	
6278	2012	90000	23.590	1364.000	94.000	5.000	0	1	
22	2015	55985	13.530	1984.000	177.010	5.000	0	0	
5925	2010	85000	17.500	1798.000	94.000	5.000	0	0	
5762	2014	51000	27.030	1969.000	190.000	5.000	0	1	



```
In [72]: #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 [73]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pred1 = linearregression.predict(X1_test)
```

```
In [74]: # Calculate MAE
mean_absolute_error(y1_test, pred1)
```

Out[74]: 1.5128627827869912

```
In [75]: #Calculate RMSE  
mean_squared_error(y1_test, pred1)**0.5
```

```
Out[75]: 3.093684570463301
```

```
In [76]: #Calculate R2 Score  
r2_score(y1_test, pred1)
```

```
Out[76]: 0.921506918435542
```

```
In [77]: #Calculate training Lr Score based on 70% training data  
linearregression.score(X1_train, y1_train)
```

```
Out[77]: 0.9450553498559194
```

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

```
In [78]: #calculate Lr score  
linearregression.score(X1_test, y1_test)
```

```
Out[78]: 0.921506918435542
```

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

In [79]: # 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_
    olsmod0 = sm.OLS(y_train, X_train1)
olsres0 = olsmod0.fit()
print(olsres0.summary())
```

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.952			
Model:	OLS	Adj. R-squared:	0.952			
Method:	Least Squares	F-statistic:	5572.			
Date:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:36:06	Log-Likelihood:	-8236.5			
No. Observations:	5075	AIC:	1.651e+04			
Df Residuals:	5056	BIC:	1.664e+04			
Df Model:	18					
Covariance Type:	nonrobust					
25	0.975]	coef	std err	t	P> t	[0.0
const	-404.8765	15.837	-25.565	0.000	-435.9	
24	-373.829	0.2014	0.008	25.530	0.000	0.1
Year	0.217	-1.62e-06	7.42e-07	-2.183	0.029	-3.08e-
86	0.165e-07	0.0435	0.006	7.141	0.000	0.0
Kilometers_Driven	0.055	0.0008	7.79e-05	10.216	0.000	0.0
06	0.001	0.0008	7.79e-05	10.216	0.000	0.0
Mileage	0.018	0.0160	0.001	16.028	0.000	0.0
32	0.018	0.0709	0.029	2.463	0.014	0.0
Power	0.127	-0.7423	0.056	-13.312	0.000	-0.8
Seats	0.127	0.5406	0.412	1.311	0.190	-0.2
14	0.349	-0.0978	0.051	-1.917	0.055	-0.1
Transmission_Manual	0.002	-0.2317	0.132	-1.754	0.079	-0.4
Owner_Type_Fourth & Above	0.027	0.2125	0.074	2.869	0.004	0.0
91	0.358	0.3899	0.066	5.909	0.000	0.2
Region_North	0.519	0.2697	0.068	3.940	0.000	0.1
67	0.404	2.6033	0.050	51.616	0.000	2.5
Region_South	0.404	8.5987	0.079	108.186	0.000	8.4
61	2.702					
Region_West	Car_Type_Level2					
35	Car_Type_Level3					

43	8.754					
Car_Type_Level4		12.0493	0.104	116.153	0.000	11.8
46	12.253					
Car_Type_Level5		10.8948	0.180	60.663	0.000	10.5
43	11.247					
Car_Type_Level6		10.5843	0.411	25.764	0.000	9.7
79	11.390					
<hr/>						
Omnibus:	358.893	Durbin-Watson:	1.998			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1743.735			
Skew:	0.123	Prob(JB):	0.00			
Kurtosis:	5.861	Cond. No.	5.85e+07			
<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 [80]:

```
#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 [81]: #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_Level12    1.586
Car_Type_Level13    2.077
Car_Type_Level14    2.577
Car_Type_Level15    1.532
Car_Type_Level16    1.107
dtype: float64
```

Get Regression Diagnostics Summary

In [82]: #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:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:36:06	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	0.1961	0.008	24.274	0.000	0.1
80	0.212	-1.718e-06	7.61e-07	-2.258	0.024	-3.21e-
06	-2.27e-07	0.0320	0.006	5.160	0.000	0.0
Mileage		0.0016	6.19e-05	25.561	0.000	0.0
20	0.044	0.0016	6.19e-05	25.561	0.000	0.0
Engine		0.0016	6.19e-05	25.561	0.000	0.0
01	0.002	0.0016	6.19e-05	25.561	0.000	0.0
Seats		-0.0793	0.028	-2.844	0.004	-0.1
34	-0.025	-0.9898	0.055	-18.024	0.000	-1.0
Transmission_Manual		-0.0778	0.052	-1.488	0.137	-0.1
97	-0.882	-0.0778	0.052	-1.488	0.137	-0.1
Owner_Type_Fourth & Above		-0.2048	0.135	-1.513	0.130	-0.4
95	1.263	0.4341	0.423	1.027	0.304	-0.3
Owner_Type_Second		0.1776	0.076	2.340	0.019	0.0
80	0.025	0.3221	0.067	4.772	0.000	0.1
Owner_Type_Third		0.2395	0.070	3.415	0.001	0.1
70	0.061	2.7833	0.050	55.226	0.000	2.6
Region_North		8.9825	0.078	115.632	0.000	8.8
29	0.326	12.7053	0.098	130.038	0.000	12.5
Region_South		11.6370	0.178	65.427	0.000	11.2
90	0.454	11.1725	0.419	26.639	0.000	10.3
Region_West						
02	0.377					
Car_Type_Level2						
85	2.882					
Car_Type_Level3						
30	9.135					
Car_Type_Level4						
14	12.897					
Car_Type_Level5						
88	11.986					
Car_Type_Level6						

```
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 [83]: `#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 [84]: #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:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:36:06	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 [85]: #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:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:36:06	Log-Likelihood:	-8288.4			
No. Observations:	5075	AIC:	1.661e+04			
Df Residuals:	5058	BIC:	1.672e+04			
Df Model:	16					
Covariance Type:	nonrobust					
25	0.975]	coef	std err	t	P> t	[0.0
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 [86]: residual= olsres3.resid
np.mean(residual)
```

```
Out[86]: -1.3657719946781153e-12
```

```
In [87]: #predicted values
residual=olsres3.resid
fitted=olsres3.fittedvalues
```

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

H₀ : Independent variable is not significant

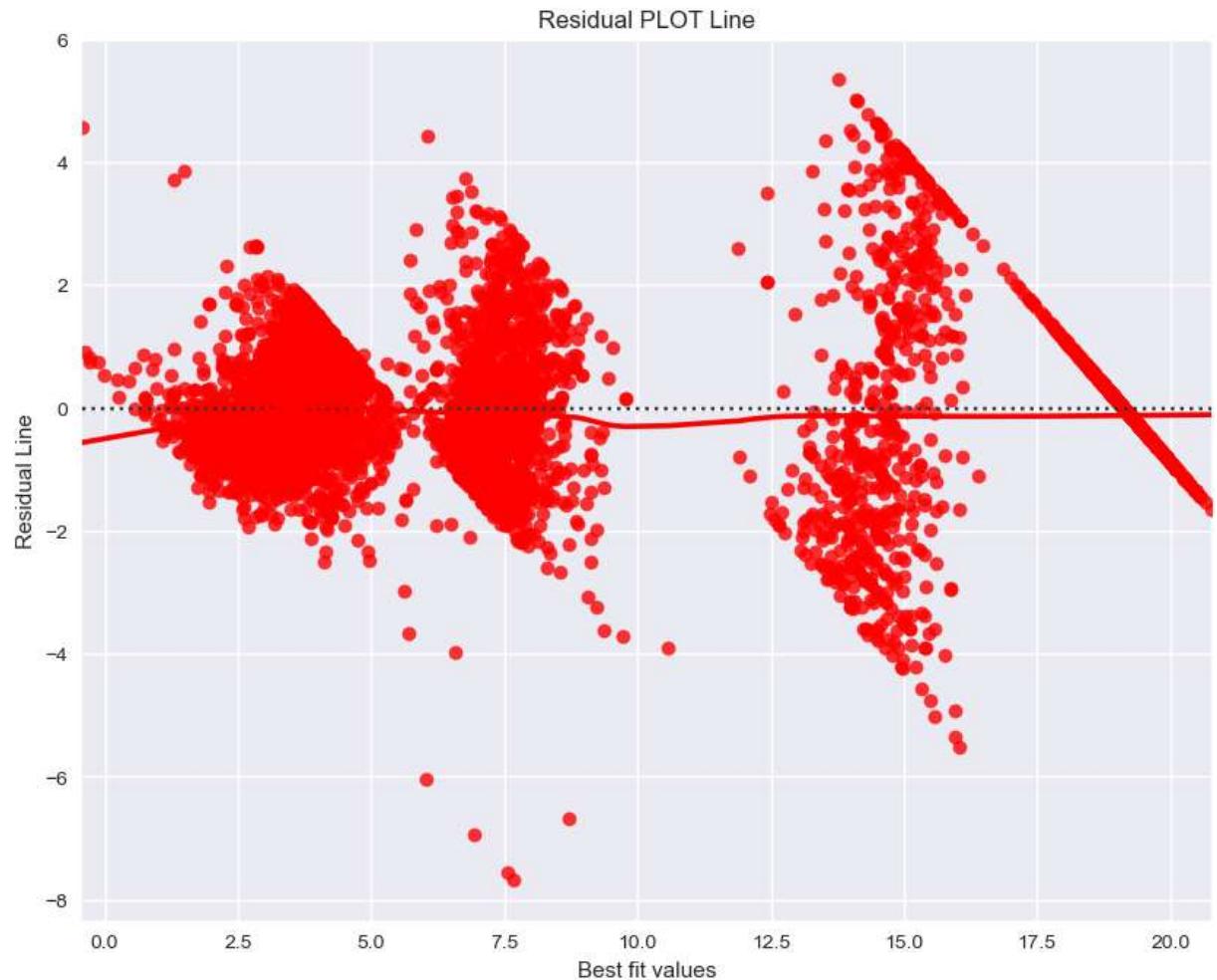
H_a : Independent variable is significant

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.

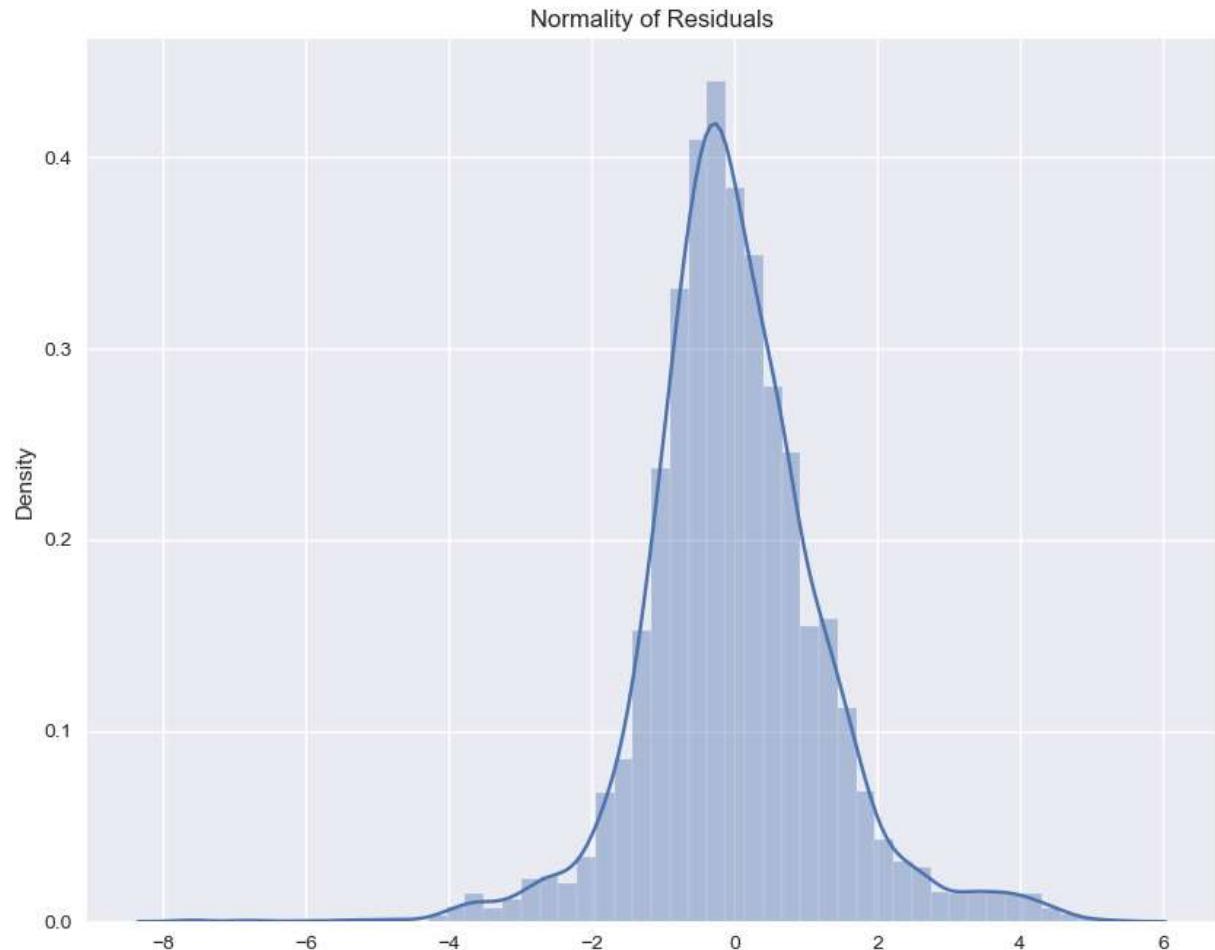
Draw a prediction line with data points on a scatter plot Showing Residuals Best Fit

```
In [88]: #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 [89]: *#Histogram of Residuals*

```
sns.distplot(residual)
plt.title('Normality of Residuals')
plt.show()
```

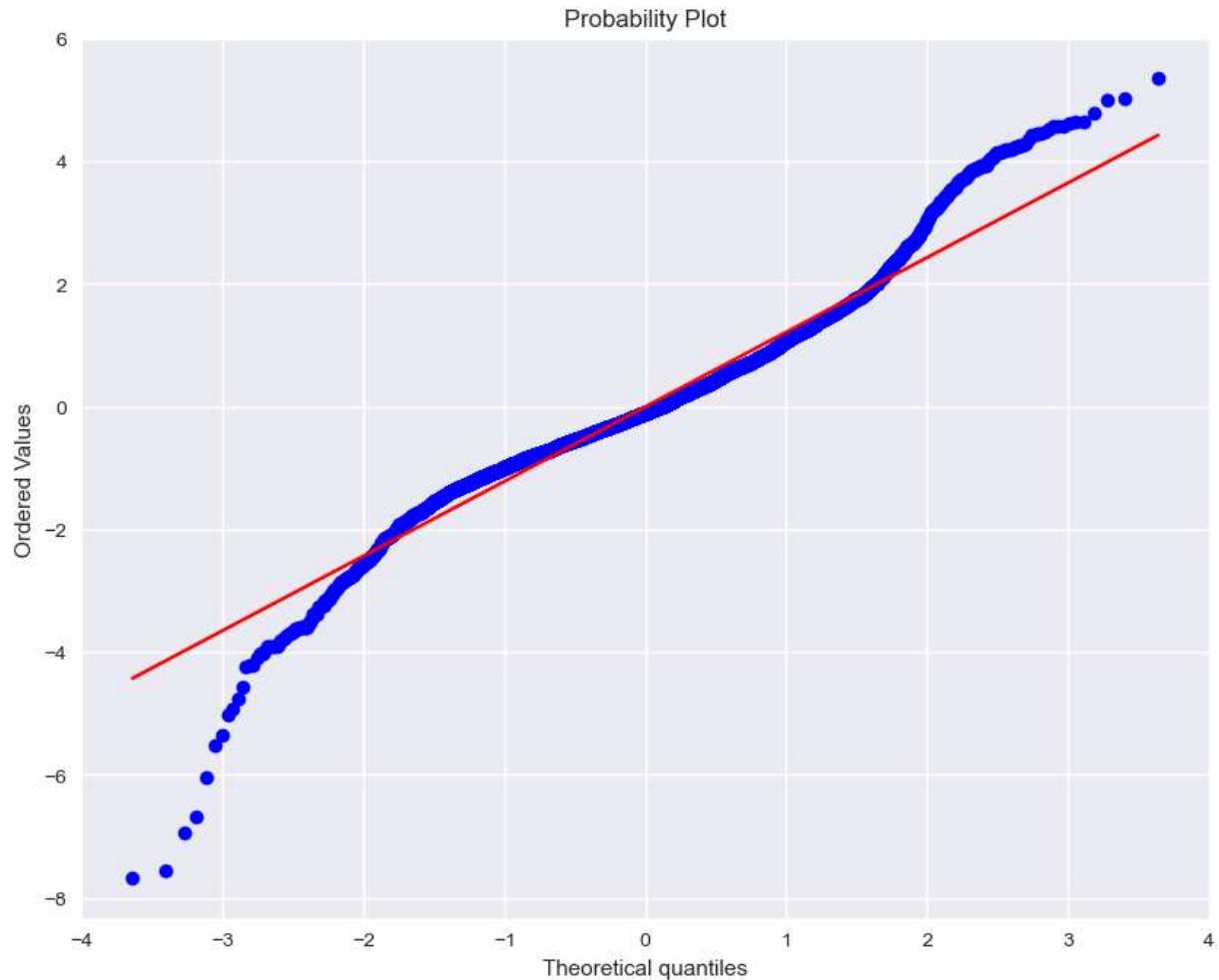


Check the normality assumptions by creating a QQ-plot

QQ Plot shows linearity see below

In [90]: # 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 [91]: #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[91]: [('F statistic', 1.033807218416428), ('p-value', 0.20199670346246976)]

Model Data for Prediction

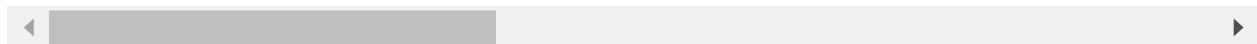
In [92]: `#Clarify x train column detail`
`X_train4.columns`

Out[92]: `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 [93]: `#Print x train head`
`X_test_final = X_test[X_train4.columns]`
`X_test_final.head()`

Out[93]:

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

Cross Validation Using Sci-kit Learn

In [95]: `#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 [96]: #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:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00			
Time:	15:36:09	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	-372.903	-415.2947	21.617	-19.212	0.000	-457.6
Year	0.227	0.2061	0.011	19.132	0.000	0.1
Mileage	0.057	0.0388	0.009	4.209	0.000	0.0
Power	0.026	0.0237	0.001	20.555	0.000	0.0
Seats	0.391	0.3157	0.038	8.271	0.000	0.2
Transmission_Manual	-0.542	-0.7067	0.084	-8.402	0.000	-0.8
Owner_Type_Fourth & Above	1.375	-0.0296	0.716	-0.041	0.967	-1.4
Owner_Type_Second	0.033	-0.1154	0.076	-1.526	0.127	-0.2
Owner_Type_Third	-0.076	-0.4693	0.201	-2.338	0.019	-0.8
Region_North	0.491	0.2735	0.111	2.468	0.014	0.0
Region_South	0.745	0.5520	0.098	5.605	0.000	0.3
Region_West	0.487	0.2863	0.102	2.803	0.005	0.0
Car_Type_Level2	2.757	2.6065	0.077	33.879	0.000	2.4
Car_Type_Level3	8.901	8.6726	0.117	74.426	0.000	8.4
Car_Type_Level4	12.399	12.0980	0.154	78.776	0.000	11.7
Car_Type_Level5	11.705	11.2200	0.247	45.392	0.000	10.7
Car_Type_Level6	13.237	11.4906	0.890	12.905	0.000	9.7

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 [97]: `#print trained OLS model
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:	Mon, 13 Jun 2022	Prob (F-statistic):	0.00						
Time:	15:36:10	Log-Likelihood:	-8288.4						
No. Observations:	5075	AIC:	1.661e+04						
Df Residuals:	5058	BIC:	1.672e+04						
Df Model:	16								
Covariance Type:	nonrobust								
25	0.975]	coef	std err	t	P> t	[0.0			
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.

Deployment:

In [98]: *## When deploying this model - ensure that the next phase has been managed appropriately
##All values have been processed and there is no need to run additional data or models
##Relay the findings directly to Used Car Lot TM for review, clarification, and action*

##Conclusion:

Main impacts to buying Used cars:

-The Ordinary Least Squares is a good model for prediction and inference at 95.1% adjusted R-squared 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 association with sale price. -When selling a used car, it is best to gauge the intentions of the purchaser and ensure that the car meets their specific criteria. -This model depicted the use of correlation coefficients and variance inflation factors to determine if the car sales price and the highest correlated variables share a direct relationship. We are not determining causation, rather correlation - only to determine what are the major determinants for the sale of the second hand car. -Variables reviewed: Power, Kilometers, Fuel, Region, Seats, and Car_Type.