An Assignment Report

*On*

# Review of Application Execution in Python: Case Study

*by*

Aditya Sharma
AU18B1009

*Under the guidance of*

Satyajit Pangaonkar

School of Engineering

Avantika University, Ujjain

**2020-2021**

**Content:** **Page**

**Introduction:**

The used car market has significantly grown in recent times, with clients ranging from used car dealers and buyers. Used cars might not be as good as new ones, but when one, running short on a budget, it can be the best option for a while. The Indian used car market is segmented by the organized and unorganized segments. However, C2C (customer to customer) channel has also been used for the sales of pre-owned cars in the market. Also, some service providers are commercially available to guide and providing a platform for the user. There are many factors that involved attention while purchasing the used car.

# <u>Assignment No-03 (CS 6301- Machine Learning)</u>

**Title:** Used Car condition prediction

**Statement:**

To create a robust model that allows stakeholders to predict the condition of a used vehicle.

## A. Identification of the Dataset :

### I. Type of the Dataset (Description) : Multivariate, Structured dataset

The dataset is related to kaggle compete dataset. It's a limited participant competition dataset. There are two datasets associated with the competition, train-data.csv and test-data.csv.

**Train-data.csv**: (Rows: 5975 , Columns: 17)

**Test-data.csv:** (Rows: 1201, Columns: 12)

### II. Data Quality and Analysis :

- ✓ Name: The brand and model of the car.
- ✓ Location: The location in which the car is being sold or is available for purchase.
- ✓ Year: The year or edition of the model.
- ✓ Kilometers_Driven: The total kilometres driven in the car by the previous owner(s) in KM.
- ✓ Fuel_Type: The type of fuel used by the car.
- ✓ Transmission: The type of transmission used by the car.
- ✓ Owner_Type: Whether the ownership is Firsthand, Second hand or other.
- ✓ Mileage: The standard mileage offered by the car company in kmpl or km/kg.
- ✓ Engine: The displacement volume of the engine in cc.
- ✓ Power: The maximum power of the engine in bhp.
- ✓ Seats: The number of seats in the car.
- ✓ New_Price: The price of a new car of the same model.
- ✓ Price: The price of the used car in INR Lakhs.

Dataset number of columns and rows.

```
In [149]: car_train.shape
Out[149]: (5975, 17)

In [150]: car_test.shape
Out[150]: (1201, 12)
```

Top 5 values from the test dataset.

```
In [7]: car_test.head()
Out[7]:
```

| | Unnamed: 0 | Name | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Mileage | Engine | Power | Seats | New_Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | Maruti Alto K10 LXI CNG | Delhi | 2014 | 40929 | CNG | Manual | First | 32.26 km/kg | 998 CC | 58.2 bhp | 4.0 | NaN |
| **1** | 1 | Maruti Alto 800 2016-2019 LXI | Coimbatore | 2013 | 54493 | Petrol | Manual | Second | 24.7 kmpl | 796 CC | 47.3 bhp | 5.0 | NaN |
| **2** | 2 | Toyota Innova Crysta Touring Sport 2.4 MT | Mumbai | 2017 | 34000 | Diesel | Manual | First | 13.68 kmpl | 2393 CC | 147.8 bhp | 7.0 | 25.27 Lakh |
| **3** | 3 | Toyota Etios Liva GD | Hyderabad | 2012 | 139000 | Diesel | Manual | First | 23.59 kmpl | 1364 CC | null bhp | 5.0 | NaN |
| **4** | 4 | Hyundai i20 Magna | Mumbai | 2014 | 29000 | Petrol | Manual | First | 18.5 kmpl | 1197 CC | 82.85 bhp | 5.0 | NaN |

Concise summary of the dataframe, data types, null value counts and column names.

```
In [8]: car_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6019 entries, 0 to 6018
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         6019 non-null   int64
 1   Name               6019 non-null   object
 2   Location           6019 non-null   object
 3   Year               6019 non-null   int64
 4   Kilometers_Driven  6019 non-null   int64
 5   Fuel_Type          6019 non-null   object
 6   Transmission       6019 non-null   object
 7   Owner_Type         6019 non-null   object
 8   Mileage            6017 non-null   object
 9   Engine             5983 non-null   object
 10  Power              5983 non-null   object
 11  Seats              5977 non-null   float64
 12  New_Price          824 non-null    object
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 658.5+ KB
```

Feature variables analysis:

```
In [10]: car_test.Mileage.min()
Out[10]: '0.0 kmpl'

In [11]: car_test.Fuel_Type.value_counts()
Out[11]: Diesel    647
         Petrol    579
         CNG         6
         LPG         2
         Name: Fuel_Type, dtype: int64
```

To view some basic statistical details like percentile, mean, std etc. of a data frame

```
In [17]: car_train.describe()
Out[17]:
```

| | Unnamed: 0 | Year | Kilometers_Driven | Seats | Price |
|---|---|---|---|---|---|
| **count** | 6019.000000 | 6019.000000 | 6.019000e+03 | 5977.000000 | 6019.000000 |
| **mean** | 3009.000000 | 2013.358199 | 5.873838e+04 | 5.278735 | 9.479468 |
| **std** | 1737.679967 | 3.269742 | 9.126884e+04 | 0.808840 | 11.187917 |
| **min** | 0.000000 | 1998.000000 | 1.710000e+02 | 0.000000 | 0.440000 |
| **25%** | 1504.500000 | 2011.000000 | 3.400000e+04 | 5.000000 | 3.500000 |
| **50%** | 3009.000000 | 2014.000000 | 5.300000e+04 | 5.000000 | 5.640000 |
| **75%** | 4513.500000 | 2016.000000 | 7.300000e+04 | 5.000000 | 9.950000 |
| **max** | 6018.000000 | 2019.000000 | 6.500000e+06 | 10.000000 | 160.000000 |

Dropping column new_price from both test and train dataset.

```
In [12]: #The new price column is missing in both train and test set, so we can remove it
         car_train=car_train.drop('New_Price',axis=1)
         car_test=car_test.drop('New_Price',axis=1)
```

Dropping NA values

```
In [23]: car_train=car_train.dropna()

In [24]: car_train.shape
Out[24]: (5975, 13)
```

The pairwise correlation of all columns in the dataframe

```
In [19]: car_train.corr()
```

Out[19]:

| | Unnamed: 0 | Year | Kilometers_Driven | Seats | Price |
|---|---|---|---|---|---|
| **Unnamed: 0** | 1.000000 | 0.002354 | -0.008734 | -0.010832 | -0.020275 |
| **Year** | 0.002354 | 1.000000 | -0.173048 | 0.012333 | 0.305327 |
| **Kilometers_Driven** | -0.008734 | -0.173048 | 1.000000 | 0.083113 | -0.011493 |
| **Seats** | -0.010832 | 0.012333 | 0.083113 | 1.000000 | 0.052225 |
| **Price** | -0.020275 | 0.305327 | -0.011493 | 0.052225 | 1.000000 |

```
In [20]: car_test.corr()
```

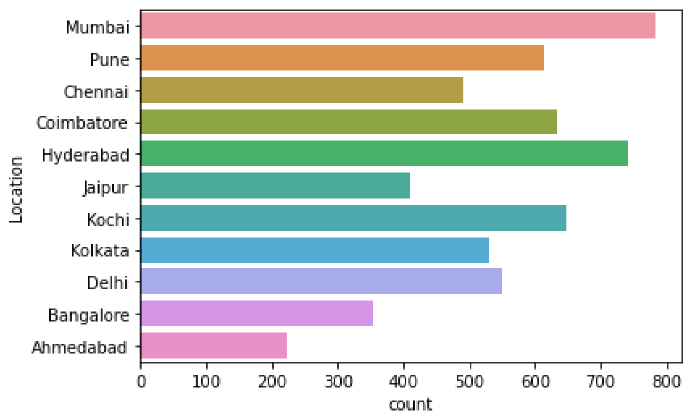
Out[20]:

| | Unnamed: 0 | Year | Kilometers_Driven | Seats |
|---|---|---|---|---|
| **Unnamed: 0** | 1.000000 | 0.027562 | -0.043032 | -0.034890 |
| **Year** | 0.027562 | 1.000000 | -0.455227 | -0.011982 |
| **Kilometers_Driven** | -0.043032 | -0.455227 | 1.000000 | 0.219258 |
| **Seats** | -0.034890 | -0.011982 | 0.219258 | 1.000000 |

Location count plot.

```
In [26]: import seaborn as sns
         sns.countplot(y='Location',data=car_train)
         print(car_train.Location.value_counts(normalize=True)*100)
```

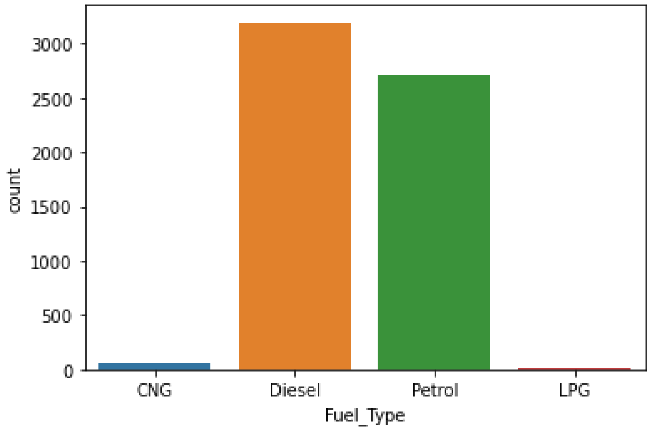
```
Mumbai        13.121339
Hyderabad     12.401674
Kochi         10.845188
Coimbatore    10.610879
Pune          10.259414
Delhi          9.188285
Kolkata        8.870293
Chennai        8.200837
Jaipur         6.861925
Bangalore      5.907950
Ahmedabad      3.732218
Name: Location, dtype: float64
```



Fuel type count plot

```
In [28]: sns.countplot(x='Fuel_Type',data=car_train)
         print(car_train.Fuel_Type.value_counts())
```

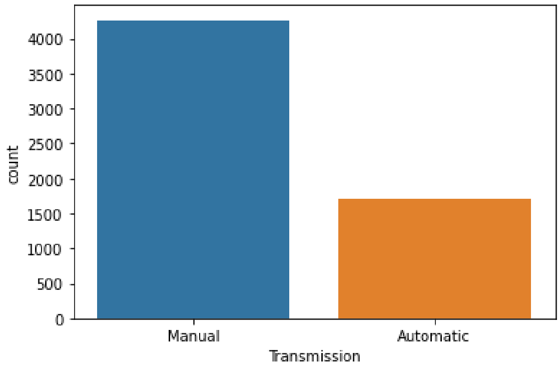
```
Diesel    3195
Petrol    2714
CNG         56
LPG         10
Name: Fuel_Type, dtype: int64
```




Transmission count plot.

```
In [29]: sns.countplot(x='Transmission',data=car_train)
         print(car_train.Transmission.value_counts(normalize=True)*100)
```

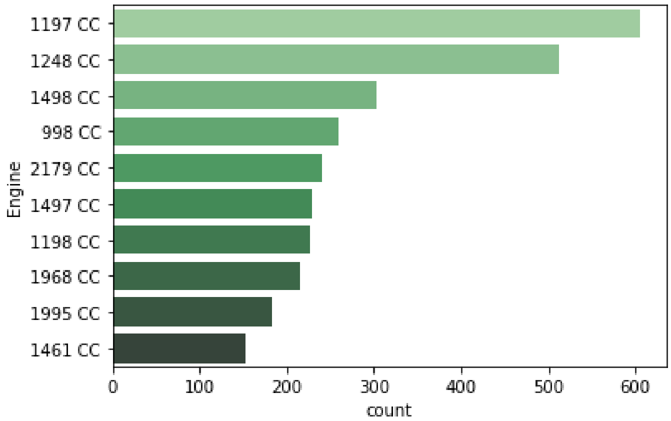
```
Manual       71.39749
Automatic    28.60251
Name: Transmission, dtype: float64
```



Engine count plot.

```
In [32]: sns.countplot(y="Engine", data=car_train, palette="Greens_d",
                       order=car_train.Engine.value_counts().iloc[:10].index)
```

```
Out[32]: <AxesSubplot:xlabel='count', ylabel='Engine'>
```




Cleaning the values in the following columns

**Power:** 58.16 bhp -> 58.16

**Milage:** 19.67 kmpl -> 19.67

**Engine:** 998 CC -> 998

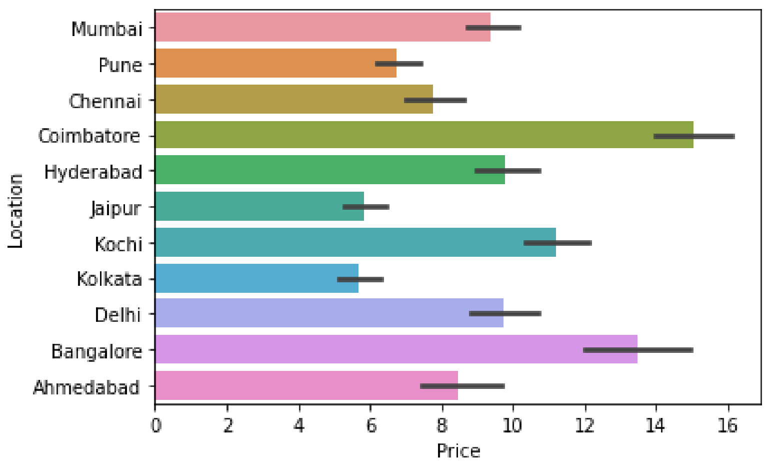
```
In [39]: car_train['power_n']=car_train.Power.str.extract(r'(\d+.\d+)').astype('float')
         car_train['milage_n']=car_train.Mileage.str.extract(r'(\d+.\d+)').astype('float')
         car_train['Engine_n']=car_train.Engine.str.extract(r'(\d+)').astype('int')
         car_train['seat_n']=car_train.Seats.astype('int')
```

**Bivatative Analysis**

Price vs Location

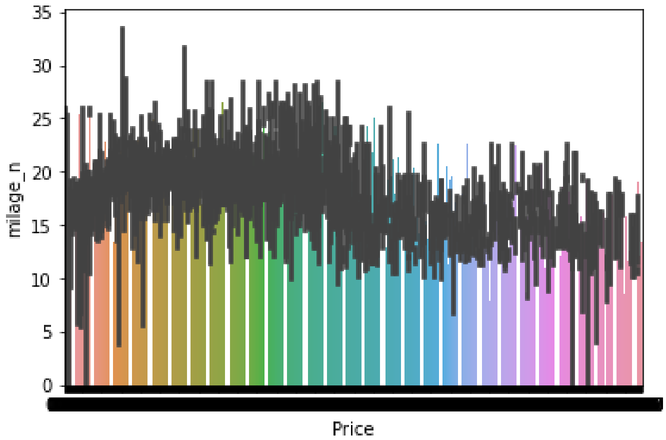
```
sns.barplot(y="Location", x="Price", data=car_train)

#here we are seeing the median price at each location
```

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

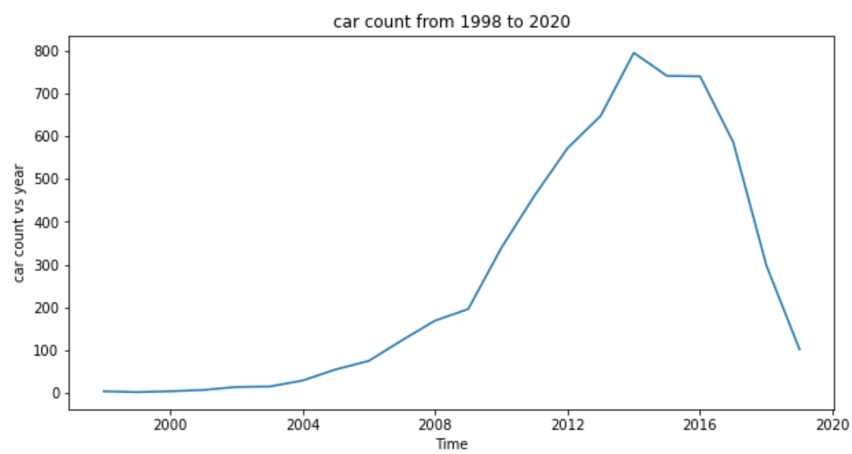


Milage vs Price

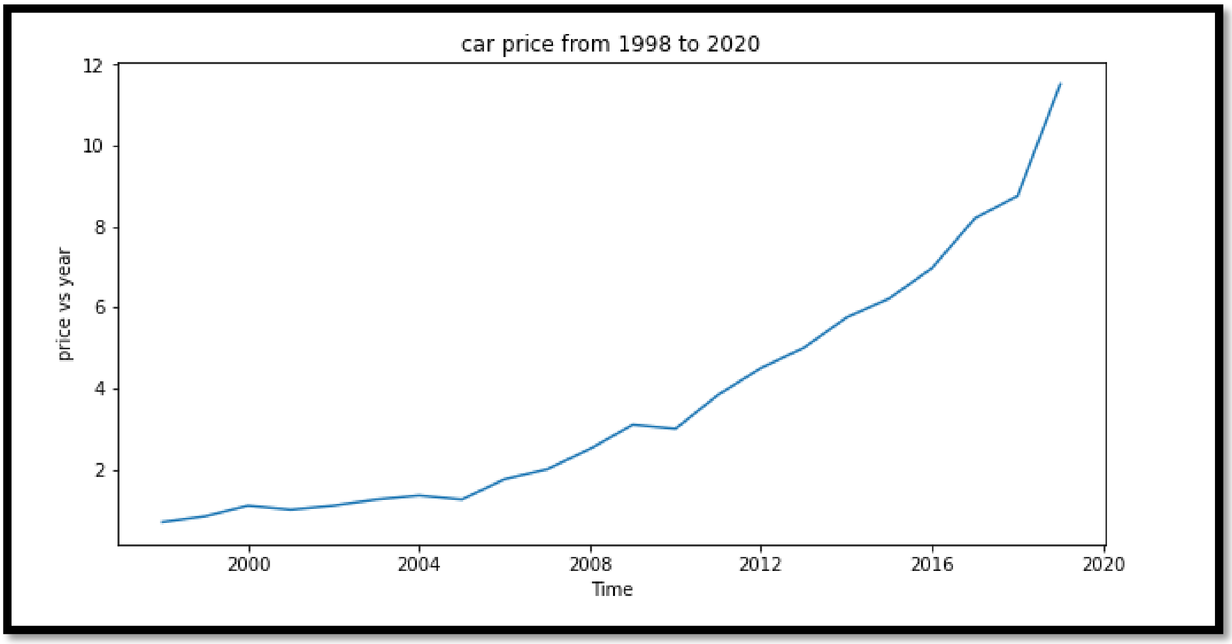
```
In [51]: #Milage vs price

         sns.barplot(y="milage_n", x="Price", data=car_train)

Out[51]: <AxesSubplot:xlabel='Price', ylabel='milage_n'>
```




Count of car vs Years



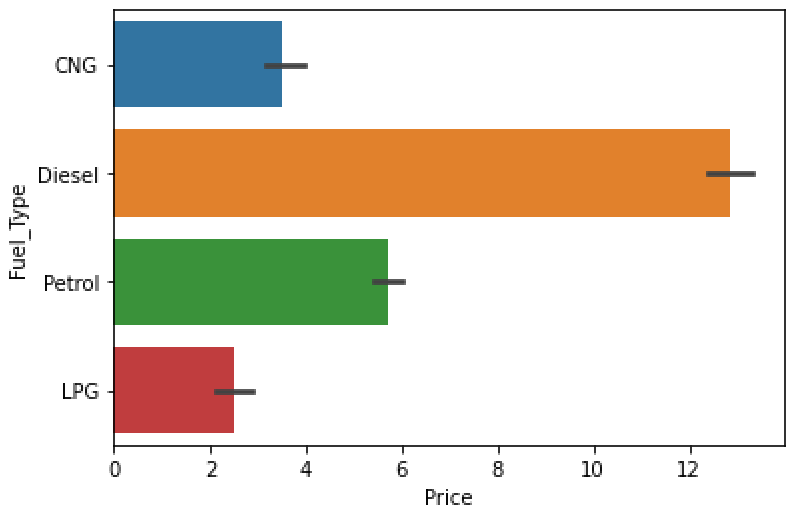
Car Price vs Years




It is been clear from the graph over the period of time the price has increased.

Fuel Type vs Price

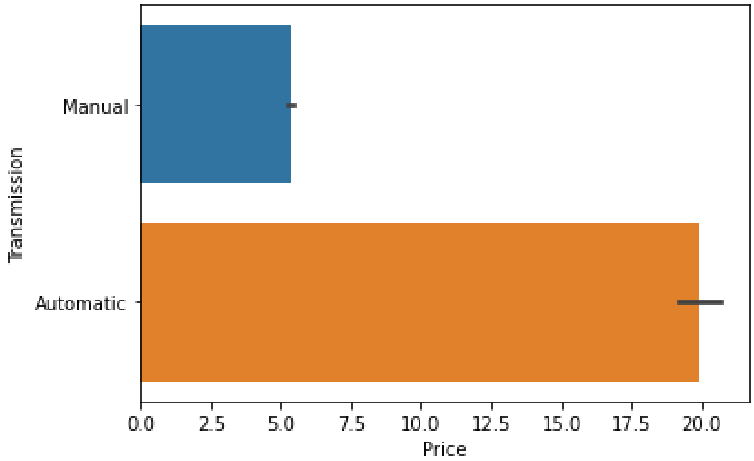
```
In [60]: # fuel type vs price
         sns.barplot(y="Fuel_Type", x="Price", data=car_train)

Out[60]: <AxesSubplot:xlabel='Price', ylabel='Fuel_Type'>
```



Transmission vs Price

```
In [61]: #Transmission vs Price
         sns.barplot(y="Transmission", x="Price", data=car_train)
```

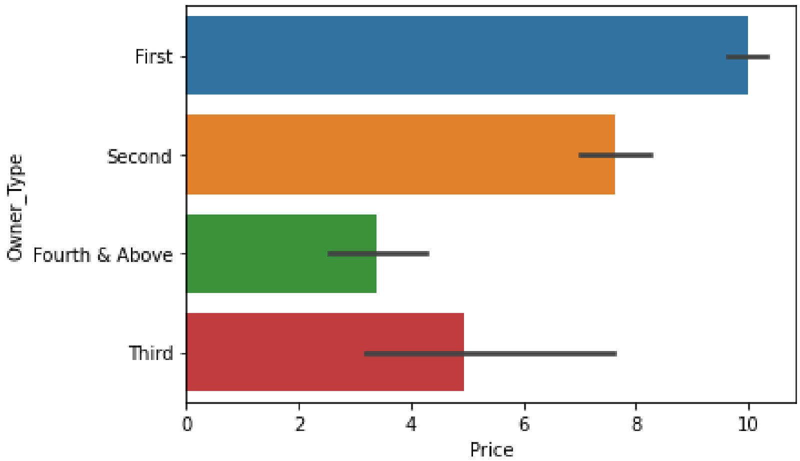
Out[61]: <AxesSubplot:xlabel='Price', ylabel='Transmission'>




Owner Type vs Price

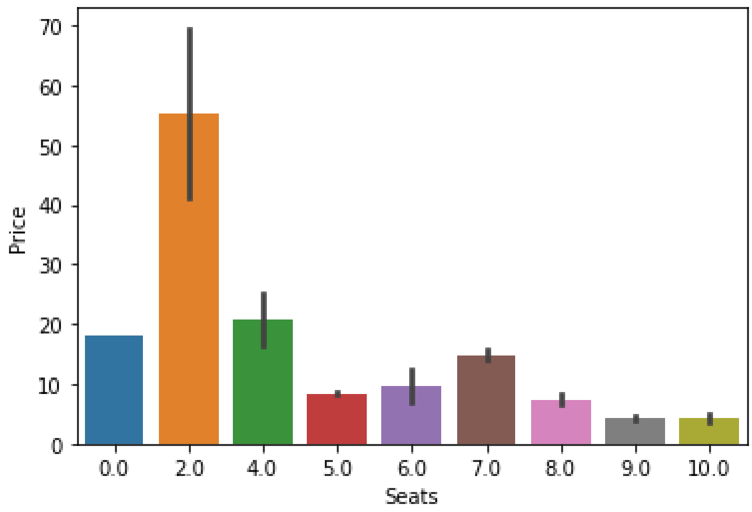
```
In [62]: #Owner_Type vs price
         sns.barplot(y="Owner_Type", x="Price", data=car_train)
```

Out[62]: <AxesSubplot:xlabel='Price', ylabel='Owner_Type'>

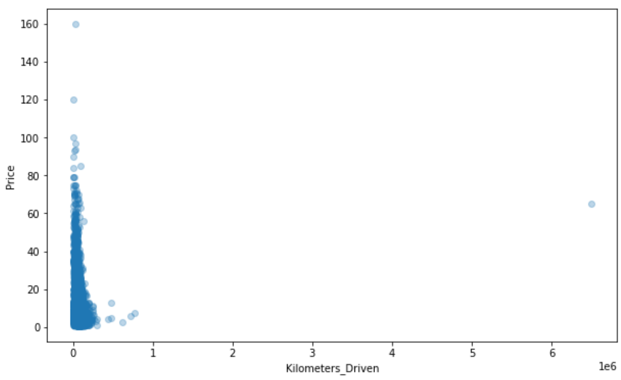


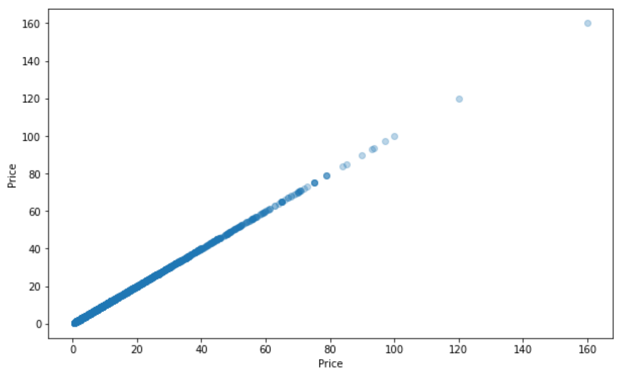
No. of seats in the car Vs Price

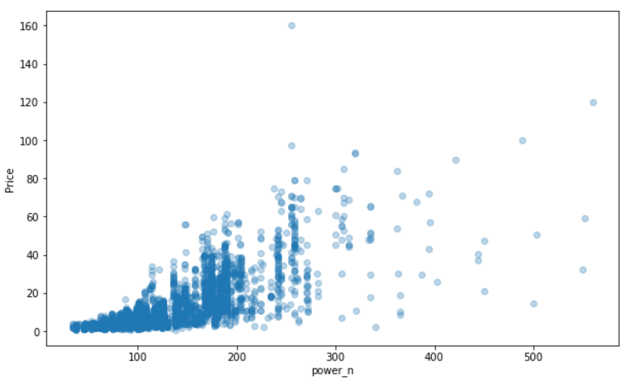
```
Out[63]: <AxesSubplot:xlabel='Seats', ylabel='Price'>
```

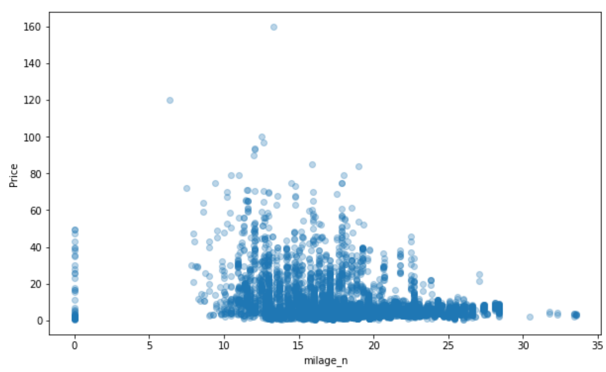



**Outliers**



Removing outliers using BaseEstimator

```
In [69]: from sklearn.base import BaseEstimator,TransformerMixin,RegressorMixin
```

```
In [70]: class RemoveOutliers(BaseEstimator,TransformerMixin):
             """This class removes outliers from data.
             """

             def fit (self,X,y=None):
                 return self


             def transform(self,X,y=None):
                 X=X[X['Kilometers_Driven']< 262000]

                 X=X[X['Price']<=100]

                 X=X[X['power_n']<= 530]

                 X=X[X['Engine_n']<= 5900 ]

                 return X
```

```
In [71]: data1=RemoveOutliers().fit_transform(car_train)
```

**Feature Transformation**

```
In [86]: class FeaturesTransformer(BaseEstimator,TransformerMixin):
             """This class trnsforms numberical featuress in the dataset.
             Transformations are hard coded.
             """
             def fit(self,X,y=None):
                 return self
             def transform(self,X,y=None):
                 import numpy as np
                 from scipy.special import boxcox1p
                 X['Kilometers_Driven']=X['Kilometers_Driven'].apply(lambda x: boxcox1p(x,0.33))

                 X['power_n']=X['power_n'].apply(lambda x:np.log(x) )

                 X['Engine_n']=X['Engine_n'].apply(lambda x: np.log(x))

                 X['milage_n']=X['milage_n'].apply(lambda x: np.log1p(x)**4)

                 return X
```

**Creating dummy variables**

Dummy Variables act as indicators of the presence or absence of a category in a Categorical Variable. The usual convention dictates that 0 represents absence while 1 represents presence. The conversion of Categorical Variables into Dummy Variables leads to the formation of the two-dimensional binary matrix where each column represents a particular category.

**One Hot Encoding.**

A one hot encoding is a representation of categorical variables as binary vectors.

This first requires that the categorical values be mapped to integer values.

Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

```
In [98]: #as Transmission  is an nominal varible lets perform onehotencoding
         Transmission = data3[["Transmission"]]

         Transmission = pd.get_dummies(Transmission,drop_first=True)

         Transmission.head()

         # Location
         location = data3[["Location"]]

         location = pd.get_dummies(location,drop_first=True)

         location.head()

         #fuel_type

         Fuel_Type = data3[["Fuel_Type"]]

         Fuel_Type = pd.get_dummies(Fuel_Type,drop_first=True)

         Fuel_Type.head()
```

Out[98]:

| | Fuel_Type_Diesel | Fuel_Type_LPG | Fuel_Type_Petrol |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 1 | 0 | 0 |
| **2** | 0 | 0 | 1 |
| **3** | 1 | 0 | 0 |
| **4** | 1 | 0 | 0 |

Concatenate dataframe >data3+ Location +  Transmission + Fuel_Type

```
In [99]: #Concatenate dataframe >data3+ company + Location +  Transmission + Fuel_Type

         data_train = pd.concat([data3 ,location ,Transmission,Fuel_Type ], axis = 1)

In [100]: data_train.head(3)
Out[100]:
```

| | Unnamed: 0 | Location | Year | Kilometers_Driven | Fuel_Type | Transmission | Owner_Type | Price | power_n | milage_n | ... | Location_Hyderabad | Location_Jaipur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Mumbai | 2010 | 118.422565 | CNG | Manual | 1 | 1.75 | 4.063198 | 121.173895 | ... | 0 | 0 |
| 1 | 1 | Pune | 2015 | 97.827204 | Diesel | Manual | 1 | 12.50 | 4.837868 | 84.142516 | ... | 0 | 0 |
| 2 | 2 | Chennai | 2011 | 101.730606 | Petrol | Manual | 1 | 4.50 | 4.485260 | 76.239001 | ... | 0 | 0 |

3 rows × 26 columns

## III. Features Pre-Processing :

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data

```
In [103]: features_c
Out[103]: ['Kilometers_Driven', 'power_n', 'milage_n', 'Engine_n']

In [104]: scaled_features = data_train.copy()

In [105]: from sklearn.preprocessing import StandardScaler

In [106]: col_names = ['Kilometers_Driven','power_n','milage_n','Engine_n']
          features = scaled_features[col_names]
          scaler = StandardScaler().fit(features.values)
          features = scaler.transform(features.values)

In [107]: scaled_features[col_names] = features
          #print(scaled_features)

In [108]: scaled_features.head(2)
Out[108]:
```

| | Unnamed: 0 | Year | Kilometers_Driven | Owner_Type | Price | power_n | milage_n | Engine_n | seat_n | Location_Bangalore | ... | Location_Hyderabad | Location_Jai |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2010 | 0.641616 | 1 | 1.75 | -1.541884 | 2.034639 | -1.354647 | 5 | 0 | ... | 0 | |
| 1 | 1 | 2015 | -0.288928 | 1 | 12.50 | 0.323319 | 0.399417 | -0.023337 | 5 | 0 | ... | 0 | |

2 rows × 23 columns

## IV. Format of the Dataset : CSV

## B. Identification of Learning Model (Supervised Learning)

I. **Algorithm used** : RandomForest

The random forest is a classification algorithm consisting of many decisions trees. **It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees** whose prediction by committee is more accurate than that of any individual tree.

II. **Methodology used** :

**One Hot Encoding.**

A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

**Creating dummy variables**

Dummy Variables act as indicators of the presence or absence of a category in a Categorical Variable. The usual convention dictates that 0 represents absence while 1 represents presence. The conversion of Categorical Variables into Dummy Variables leads to the formation of the two-dimensional binary matrix where each column represents a particular category.

III. **Model building, Training & Testing** :

### Model Training

```
In [122]: y=scaled_features['Price']
          X=scaled_features.drop('Price',axis=1)

In [123]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 25)
```

**Linear Regression**

# Linear Regression

```
In [127]: from sklearn.linear_model import LinearRegression
          linear_reg = LinearRegression()
          linear_reg.fit(X_train, y_train)
          y_pred= linear_reg.predict(X_test)
          print("Accuracy on Traing set: ",linear_reg.score(X_train,y_train)*100,"%")
          print("Accuracy on Testing set: ",linear_reg.score(X_test,y_test)*100,"%")

          Accuracy on Traing set:  66.48679499628179 %
          Accuracy on Testing set:  67.8906639583188 %
```

**Random Forest**

# RandonForest

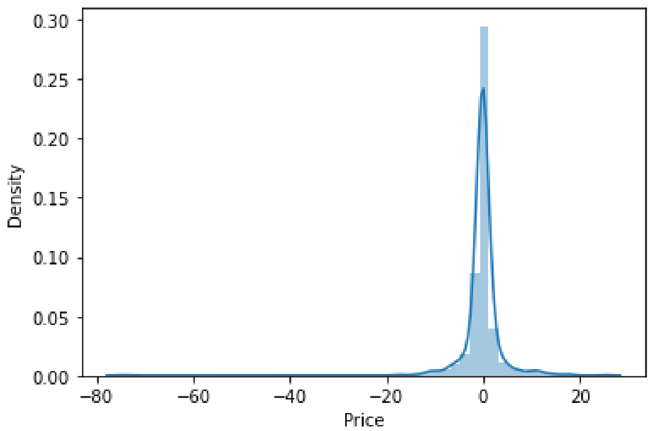
```
In [128]: from sklearn.ensemble import RandomForestRegressor
          reg_rf = RandomForestRegressor()
          reg_rf.fit(X_train, y_train)
          y_pred= reg_rf.predict(X_test)
          print("Accuracy on Traing set: ",reg_rf.score(X_train,y_train)*100,"%")
          print("Accuracy on Testing set: ",reg_rf.score(X_test,y_test)*100,"%")

          Accuracy on Traing set:  98.58226128836158 %
          Accuracy on Testing set:  86.73912912305634 %
```

## IV. Model Accuracy, Prediction & Precession :

```
In [129]: y_pred = reg_rf.predict(X_test)

In [136]: from sklearn import metrics
          from sklearn.metrics import mean_squared_error, mean_absolute_error

          print("\t\tError Table")
          print('Mean Absolute Error      : ', metrics.mean_absolute_error(y_test, y_pred))
          print('Mean Squared  Error      : ', metrics.mean_squared_error(y_test, y_pred))
          print('Root Mean Squared  Error : ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
          print('R Squared Error          : ', metrics.r2_score(y_test, y_pred)*100, '%')

                          Error Table
          Mean Absolute Error      :  1.8073331632653065
          Mean Squared  Error      :  16.781610776030604
          Root Mean Squared  Error :  4.09653643655596
          R Squared Error          :  86.73912912305634 %
```

In [131]:
```
sns.distplot(y_test-y_pred)
plt.show()
```

```
C:\Users\Hp\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureW
ill be removed in a future version. Please adapt your code to use either `disp
ility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



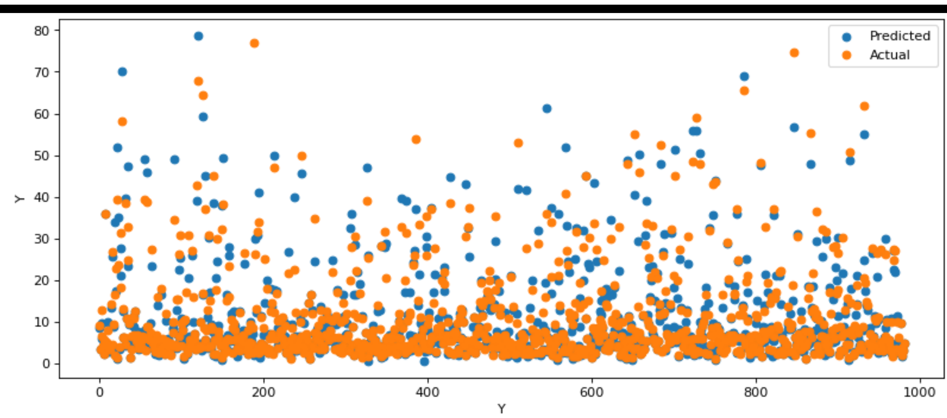

In [132]:
```
from sklearn import metrics
y_pred = reg_rf.predict(X_test)
print('Accuracy Score:')
print(metrics.r2_score(y_test, y_pred)*100,"%")
```

```
Accuracy Score:
86.73912912305634 %
```

In [133]:
```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared Error          : ', metrics.r2_score(y_test, y_pred)*100, "%")

print("Accuracy on Traing set: ",reg_rf.score(X_train,y_train)*100, "%")
```

```
MAE: 1.8073331632653065
MSE: 16.781610776030604
RMSE: 4.09653643655596
R Squared Error          :  86.73912912305634 %
Accuracy on Traing set:  98.58226128836158 %
```

Predicted value vs Actual Value



## C. Key Learning Outcomes :

After the completing this assignment I came to know that the data cleaning and standardization plays an important role for the accuracy of the model. Data cleaning, filling null, data standardization of values can abruptly increase the accuracy of the model. For the above case study, I had designed two models, one without standardization of the features, another one with standardisation of features and converting the objects to categorical variables using one hot encoding, the second model proved to be more accurate and precise as compared to the one without standardization and categorical conversion.

**Viva Questions:**

**Random Forest and Linear Regression Accuracy**

In the dataset we have 4-5 features which are majorly effecting the price of the car, also dataset contains features some of which are Categorical Variables and some of the others are continuous variable. Decision Tree is better than **Linear Regression**, since Trees can accurately divide the data based on Categorical Variables. Decision Trees are great for obtaining non-linear relationships between input features and the target variable.

Random Forest can handle messier data and messier relationships better than regression models. The averaging of the predicted values makes Random Forest better than a single Decision Tree hence improves its accuracy and reduces overfitting.

**Random Forest and K-Means Clustering (Assigning value of k in Random Forest?)**

The random forest algorithm is a supervised learing model; it uses labeled data to "learn" how to classify unlabeled data. This is the opposite of the K-means Cluster algorithm, it is an unsupervised learning model. The Random Forest Algorithm is used to solve both regression and classification problems, making it a diverse model that is widely used by engineers.

The k-means clustering algorithm assigns data points to categories, or clusters, by finding the mean distance between data points. It then iterates through this technique in order to perform more accurate classifications over time. Since you must first start by classifying your data into k categories.

**Submitted By:**

Aditya Sharma

AU18B1009