MACHINE LEARNING – 18CSC311

Used Car prediction

Case study:

Name	Roll Number
Krithiga M	CB.SC.I5DAS19020
Smrithi N	CB.SC.I5DAS19029
Varun Kumar M	CB.SC.I5DAS19032
Chinmaya Y	CB.SC.I5DAS19033

Table of content:

Abstract:

Determining whether the listed price of a used car is worth it, is a challenging task, due to the many factors that drive a used vehicle's price on the market. The focus of this project is developing machine learning models that can accurately predict the price of a used car based on its features, to make informed purchases.

Problem Statement:

The prices of new cars in the industry are fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthy. But due to the increased price of new cars and the incapability of customers to buy new cars due to the lack of funds, used cars sales are on a global increase. There is a need for a used car price prediction system to effectively determine the worthiness of the car using a variety of features. Even though there are websites that offers this service, their prediction method may not be the best. Besides, different models and systems may contribute on predicting power for a used car's actual market value. It is important to know their actual market value while both buying and selling.

Motivation:

All the people who start off with their jobs always have one of the two dreams or both the dreams of building their own house or buying their own transport especially a car. For helping people with the dream of getting their own vehicles come true, we built a machine to predict the price of the used car accurately based on a few factors.

Challenges:

Our ultimate challenge is to predict an accurate price estimate based on the description of the car. A genuine price must be suggested to the buyer and the sellers to be mutually benefitted. The price given by the sellers may not be accurate as there are a lot of external factors that influence the price. For the customers to get a proper estimate of the price machine learning Is introduced in this problem.

Requirements:

- Python – Packages:
  - numpy
  - pandas
  - seaborn
  - matplotlib
  - scikit

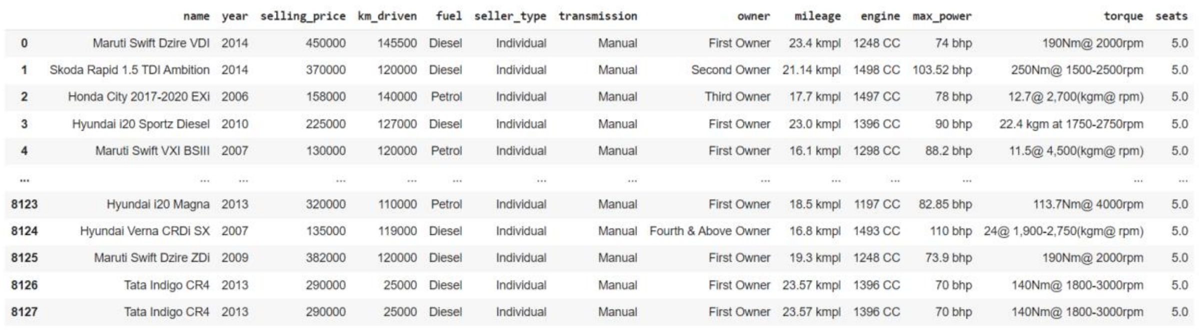
About Data:

The dataset contains 14 columns and 6019 rows. The price of the used car mainly depends upon the kilometers driven, year and owner type. Mileage of the car shows the efficiency of the car. The dataset contains all the above information along with the name of the car, location, fuel type, transmission, engine CC, horsepower, seats, new price, price.

No. of rows = 6019

No. of columns = 14

Data:



	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl	1197 CC	82.85 bhp	113.7Nm@ 4000rpm	5.0
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmpl	1493 CC	110 bhp	24@ 1,900-2,750(kgm@ rpm)	5.0
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.3 kmpl	1248 CC	73.9 bhp	190Nm@ 2000rpm	5.0
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl	1396 CC	70 bhp	140Nm@ 1800-3000rpm	5.0

Removing units from milage, engine, max_power and torque.

Nature of Output:

Numeric value of best car price based on its conditions or customer preferences.

Data Pre-Processing:

1. **Duplicate Handling:**

   Sum of duplicates: 1202

   Shape of data before handling duplicates: (8128,13)

   Shape of data after handling duplicates: (6926,13)

   Duplicate values increase the accuracy of model. But we need to find good model suitable for all conditions. With duplicates it will be overfitting.

2. **Null value Handling:**

   Number of Null cells: 1039

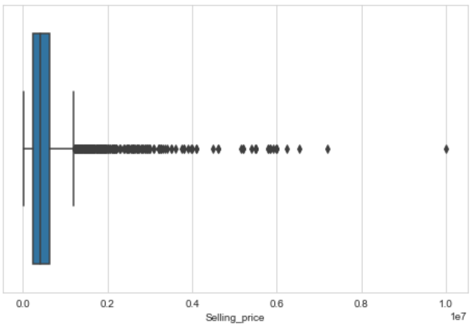
   Null values mean missing values. It exists when data of cell is not available. We can either remove complete row or handle null values. We can't remove all rows as it reduces the size of data as well as accuracy of model. So, it is better to use different measures to fill null values.

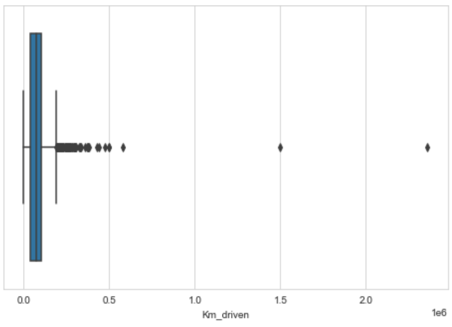
Replacing the Milage(kmpl) and Max power(bhp) with mean

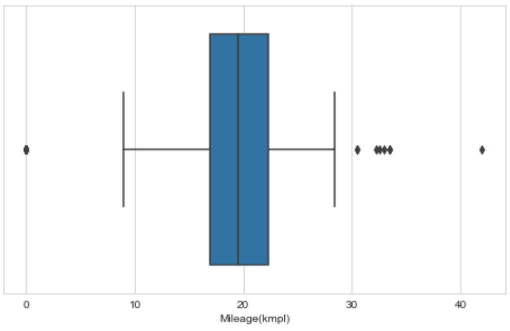
Replacing number of seats with mode as well as removing torque.

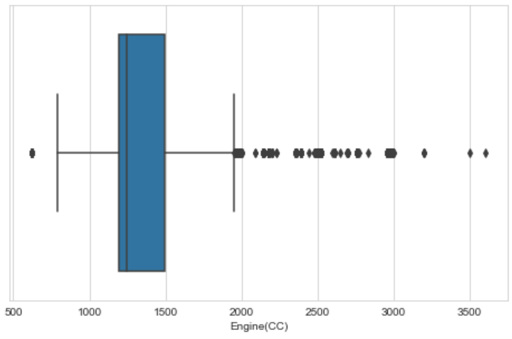
3. Outlier Treatment:

Before treatment

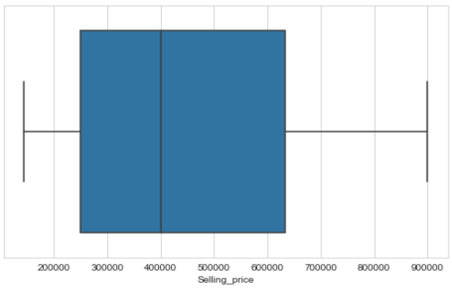


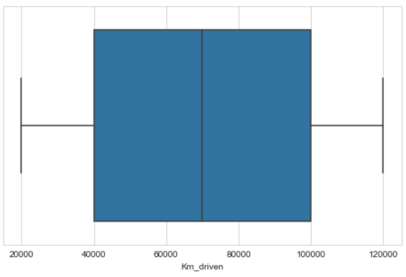


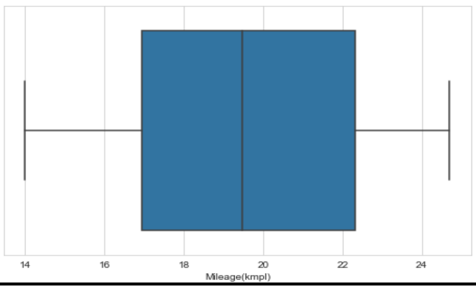


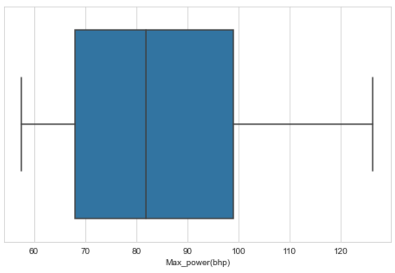


After treatment





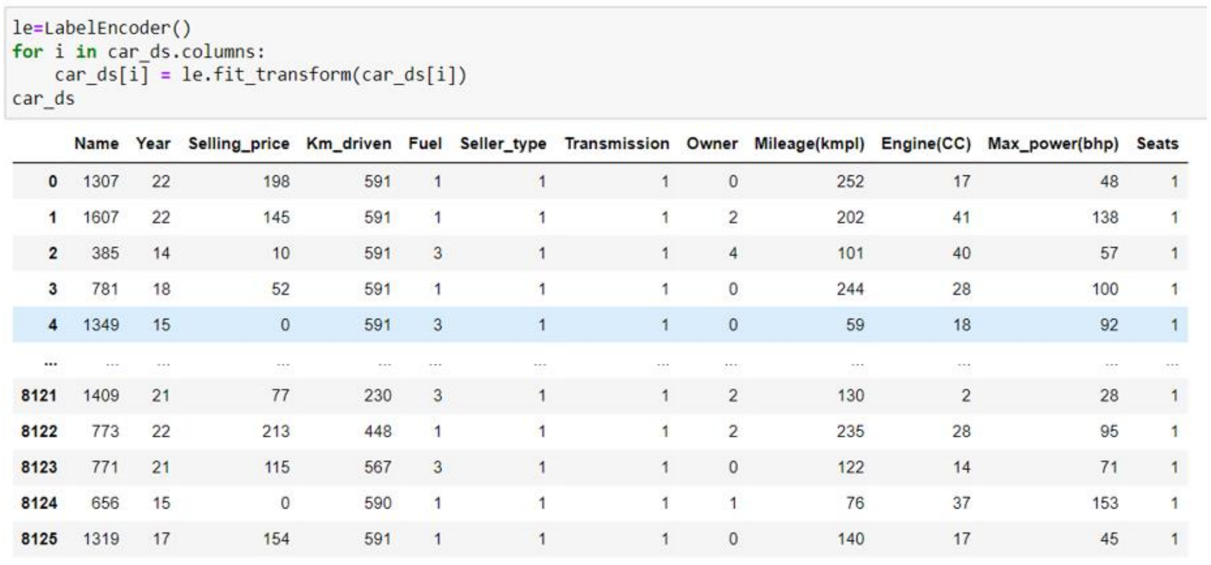




Outliers badly affect mean and standard deviation of the dataset. These may statistically give erroneous results. It increases the error variance and reduces the power of statistical tests. So, outlier should be avoided to increase the accuracy of model.

4. Encoding Variable:

Conversion of categorical variable into numerical variable so that it can be used in exploratory data analysis. Then we can get relationship between other attributes.



```
le=LabelEncoder()
for i in car_ds.columns:
    car_ds[i] = le.fit_transform(car_ds[i])
car_ds
```

	Name	Year	Selling_price	Km_driven	Fuel	Seller_type	Transmission	Owner	Mileage(kmpl)	Engine(CC)	Max_power(bhp)	Seats
0	1307	22	198	591	1	1	1	0	252	17	48	1
1	1607	22	145	591	1	1	1	2	202	41	138	1
2	385	14	10	591	3	1	1	4	101	40	57	1
3	781	18	52	591	1	1	1	0	244	28	100	1
4	1349	15	0	591	3	1	1	0	59	18	92	1
...	...	...	...	...	...	...	...	...	...	...	...	...
8121	1409	21	77	230	3	1	1	2	130	2	28	1
8122	773	22	213	448	1	1	1	2	235	28	95	1
8123	771	21	115	567	3	1	1	0	122	14	71	1
8124	656	15	0	590	1	1	1	1	76	37	153	1
8125	1319	17	154	591	1	1	1	0	140	17	45	1

5. Normalization:

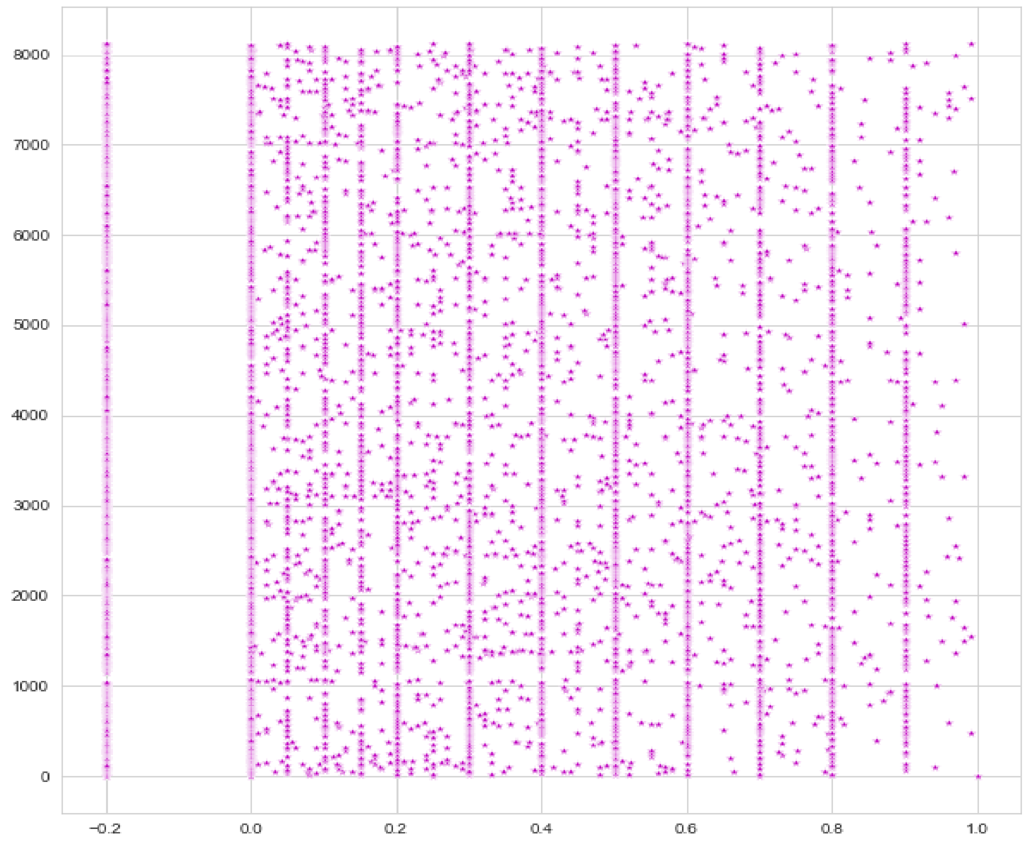
Normalization gives equal weights/importance to each variable so that no single variable steers model performance in one direction just because they are bigger numbers. So, we used min-max scaler so that we can fix same interval in [0,1] or [-1,1] if there are negative values in dataset.

```python
def minmax_scale(attribute):

    x_min = min(attribute,key=lambda x:float(x))
    x_max = max(attribute,key=lambda x:float(x))

    norm=[]
    for i in attribute:
        x_dash=(i-x_min)/(x_max-x_min)
        norm.append(x_dash)
        attribute.replace(i,x_dash,inplace=True)

    print("Some normalized values: ",norm[:10])
    print(" ")

    print("Scatterplot of normalized values:")

    fig=plt.figure(figsize=(10,10))
    sns.scatterplot(x=norm,y=car_ds.index,color='m',marker='*')
```

NORMALIZATION OF SELLING_PRICE

```python
minmax_scale(car_norm['Selling_price'])
```

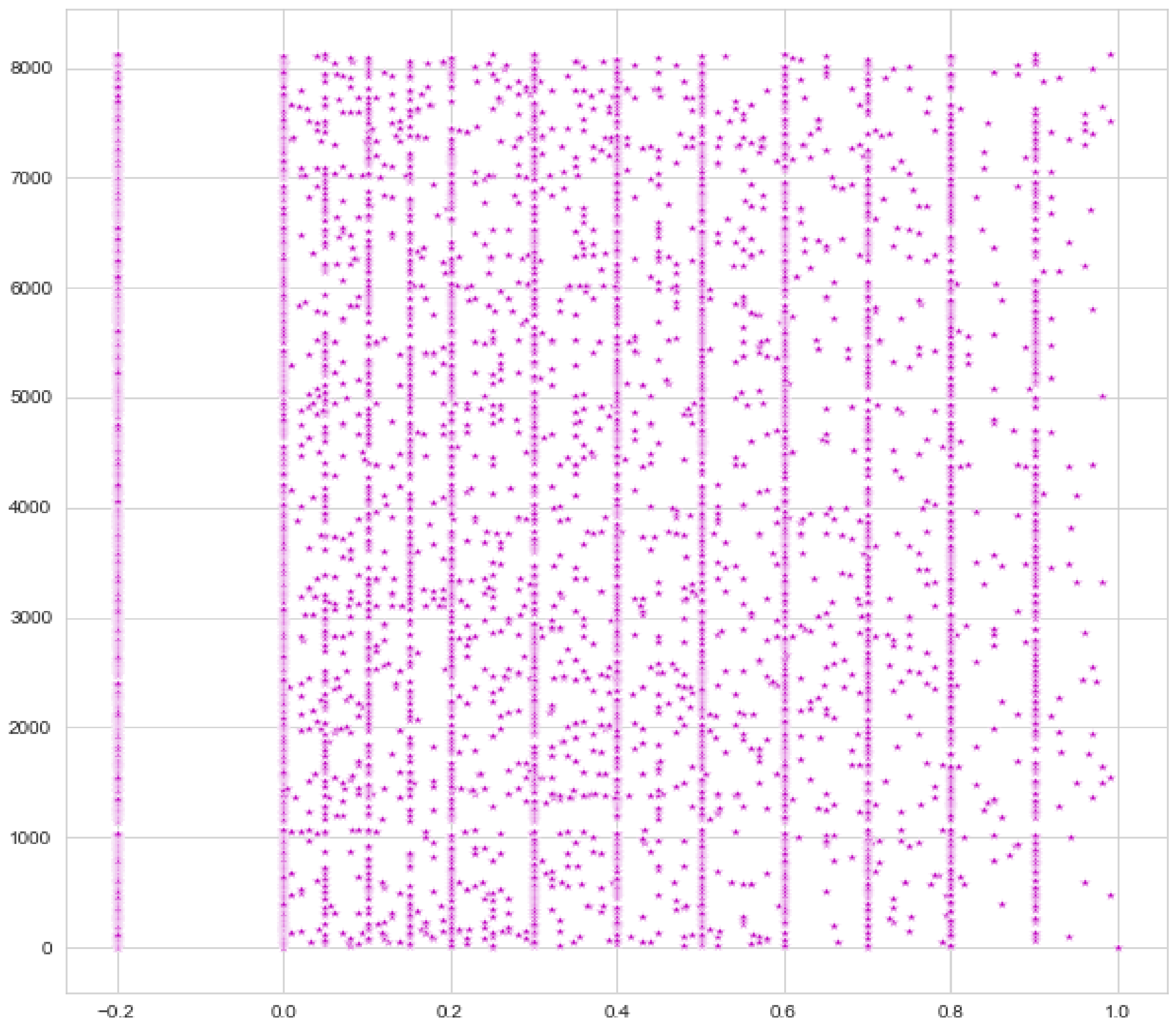
```
Some normalized values:  [0.40397350993377484, 0.2980132450331126, 0.017218543046357615, 0.10596026490066225, 0.0, 0.39072847682119205, 0.0, 0.0, 0.271523178807947, 0.0728476821192053]
```



NORMALIZATION OF KM_DRIVEN

```python
minmax_scale(car_norm['Km_driven'])
```

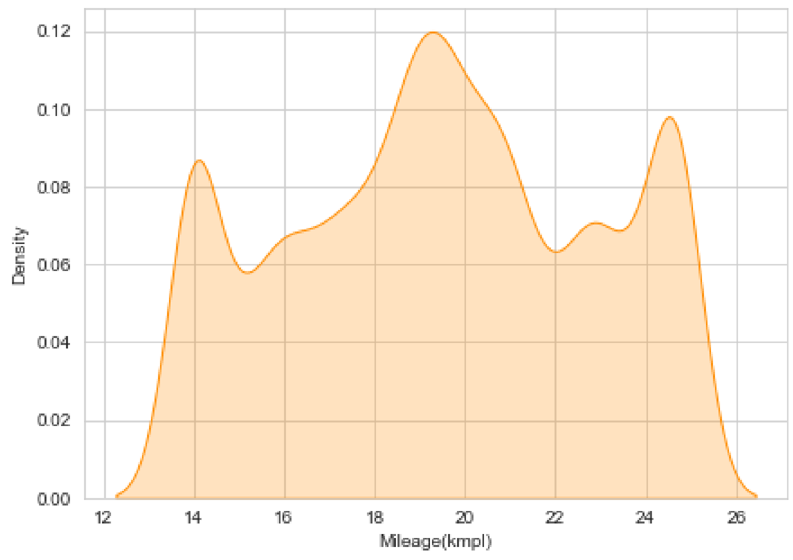
```
Some normalized values:  [1.0, -0.19999, -0.19999, -0.19999, -0.19999, 0.25, -0.19999, 0.0, 0.7, -0.19999]
```



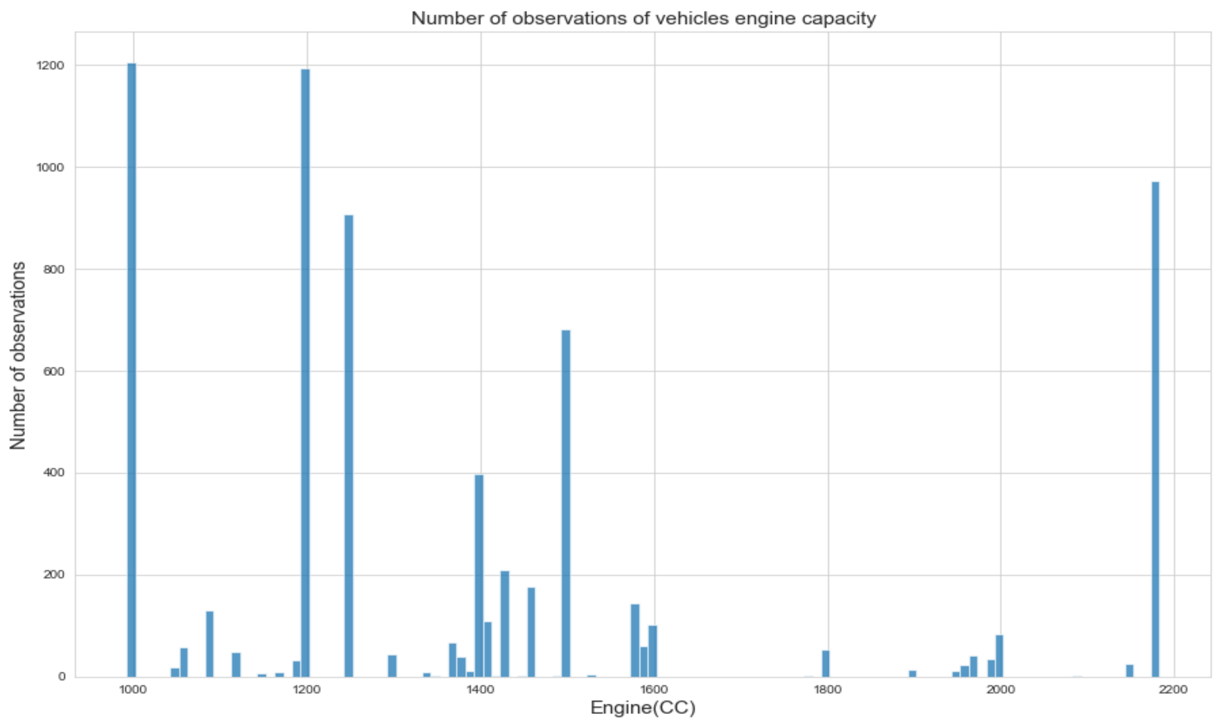
6. Exploratory Data Analysis:

Exploratory Data Analysis refers to the critical process of performing initial investigations on data to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

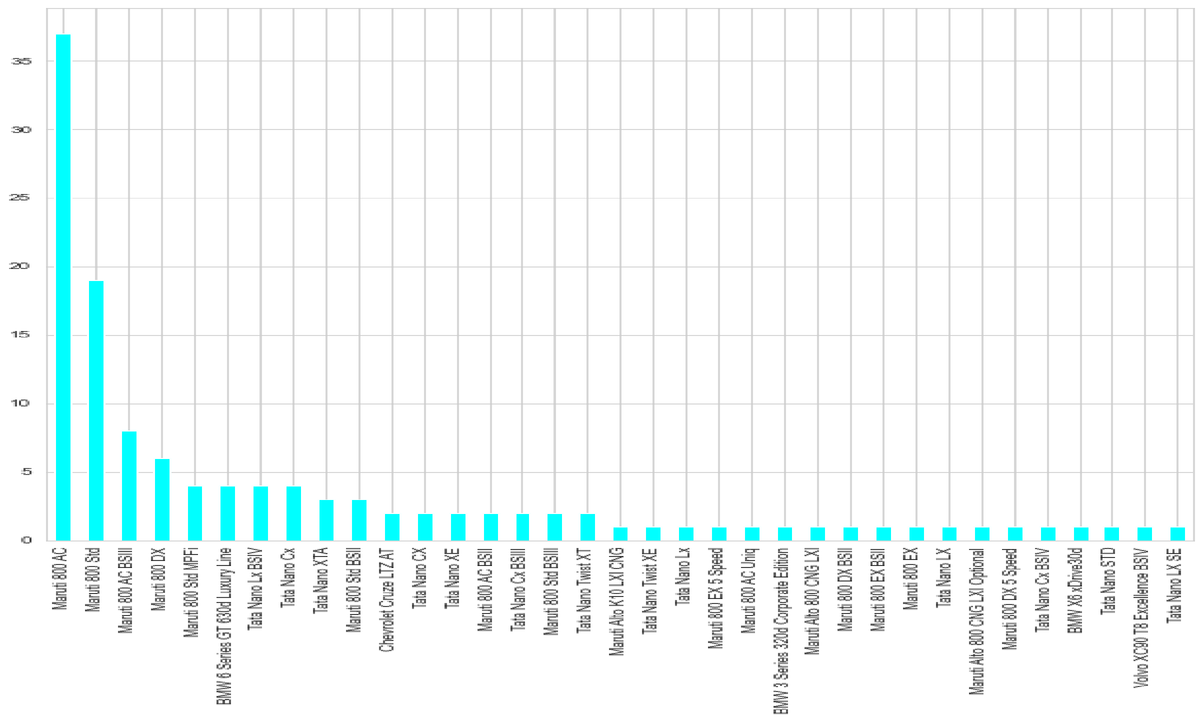
Univariate Analysis:



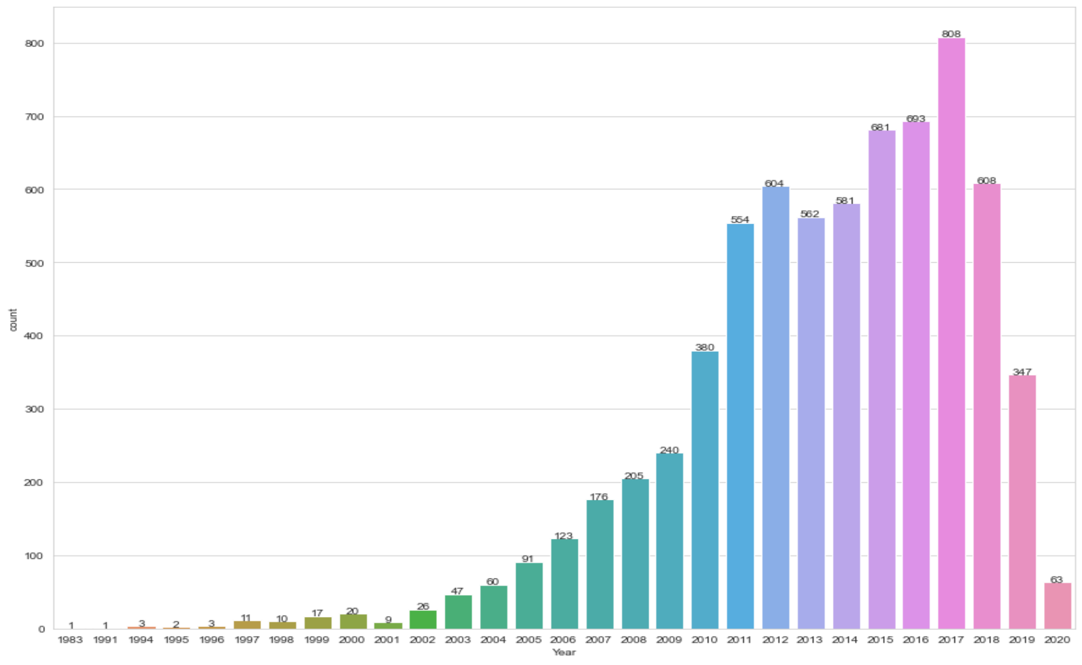
The above Kdeplot shows that the probability distribution is higher in the range of 18kmpl to 20 kmpl mileage. So many used vehicles have the mileage in this range.



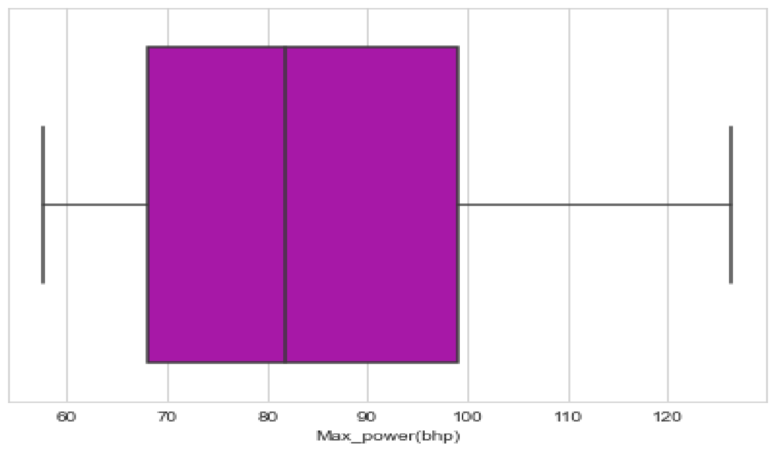
The above histogram shows that most of the used vehicles have an engine CC of 1000 and least number of vehicles have an engine CC of around 1500.



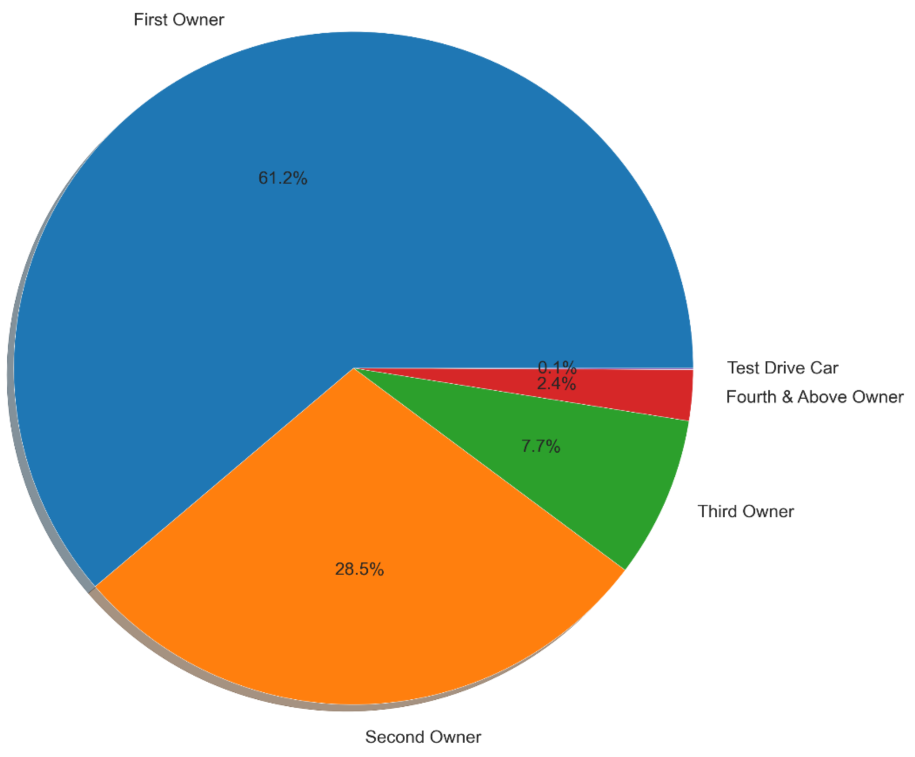
The above barplot shows that most number of Maruti 800 AC vehicles have 4 seats



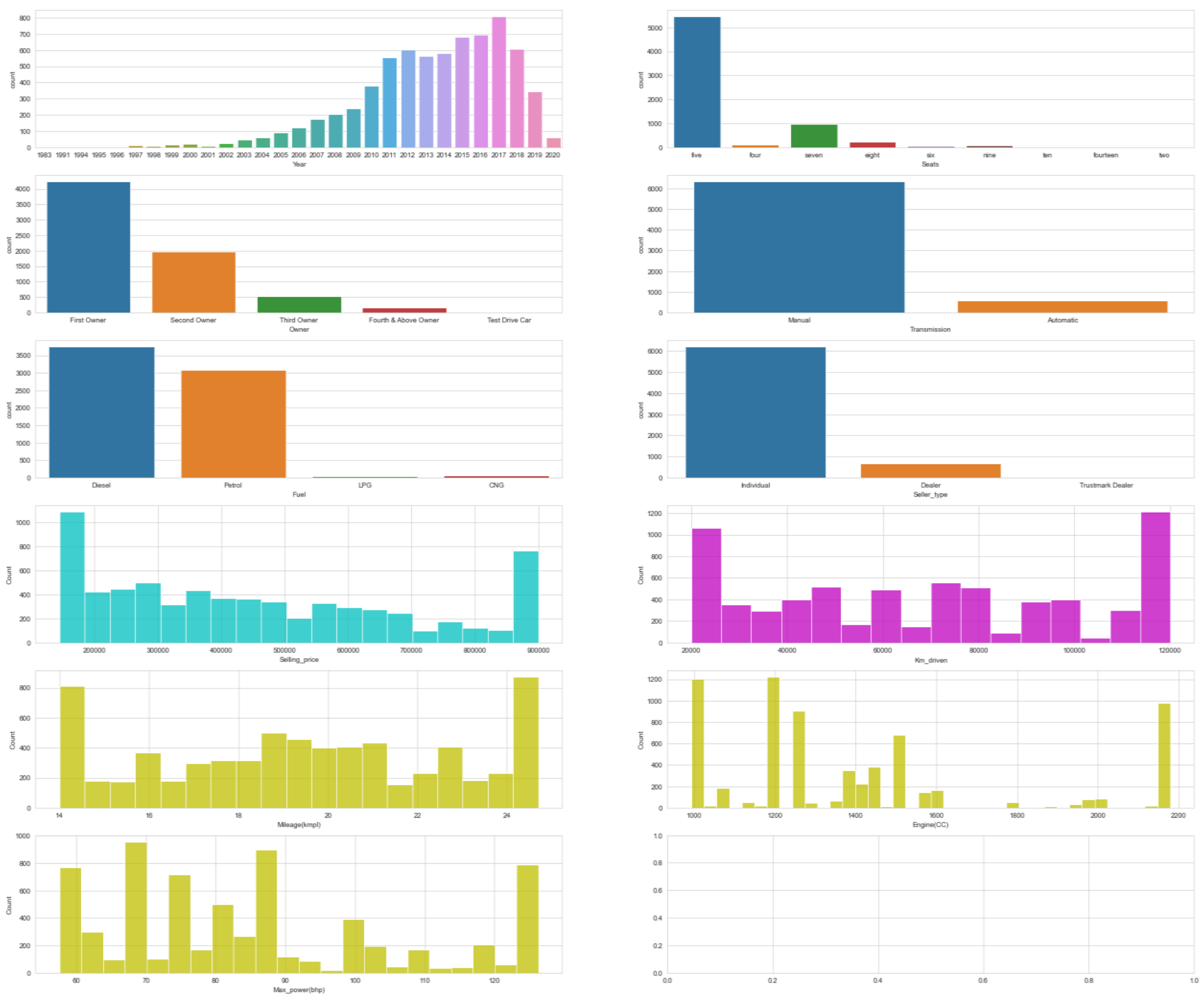
The above counplot shows that most vehicles were bought in the year 2017 and least number of vehicles were bought during 1983 and 1991



The above boxplot shows that the minimum max power lies below 60bhp and maximum max power lies above 120bhp.



Most of the used vehicles belong to their first owner and least number of vehicles are used as test drive car



-->The first countplot shows that in the year 2017 most vehicles were bought and in the year 1983 and 1991 least number of vehicles were bought.

-->The second countplot shows that most vehicles have five seats while least number of vehicles have fourteen seats.

-->The third countplot shows that most vehicles are owned by first owners while least number of vehicles are used for test drive.

-->The fourth countplot shows that most vehicles are of manual transmission.

-->The fifth counplot shows that most vehciles use diesel as their fuel and very few vehicles use LPG as their fuel.

-->The sixth counplot shows that most vehicles are sold by individuals than dealers and trustmark dealers.

-->The seventh histplot shows that most vehicles have a selling price in the range of 2 lakhs while least vehicles have a selling price of 7 lakshs.

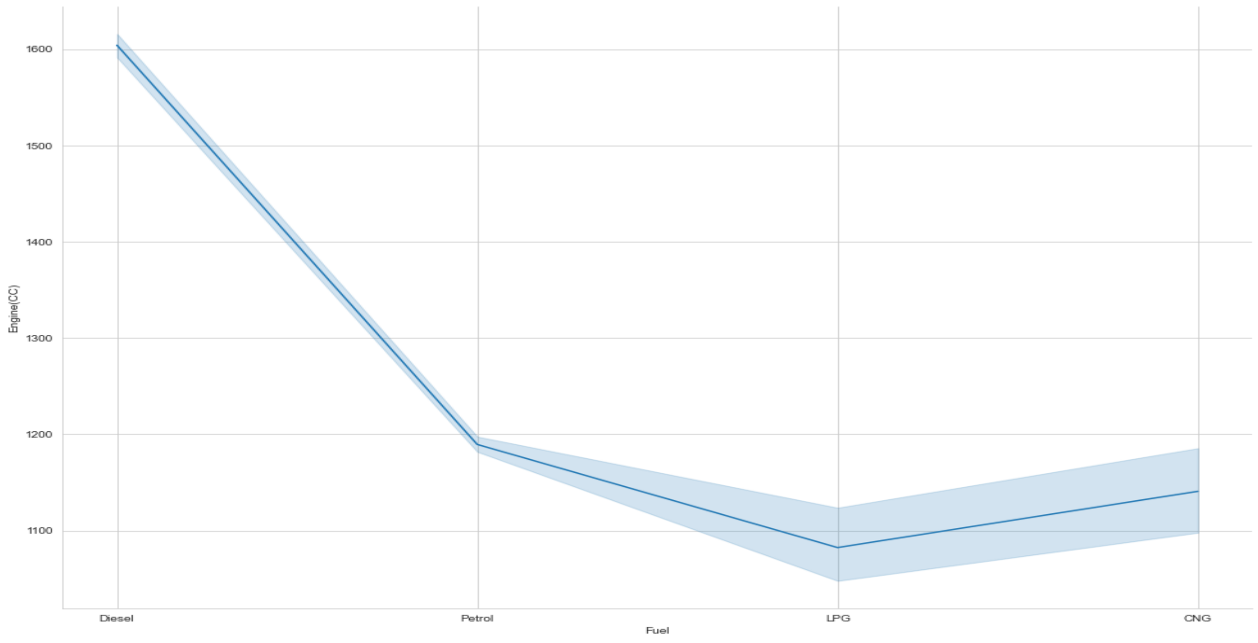
-->The eighth histplot shows that most vehicles have driven around 1,20,000 kms while the least number of vehicles have driven around 1 lakh kms.

-->The nineth histplot shows that most vehicles have mileage around 24kmpl and least vehicles have mileage around 22kmpl.

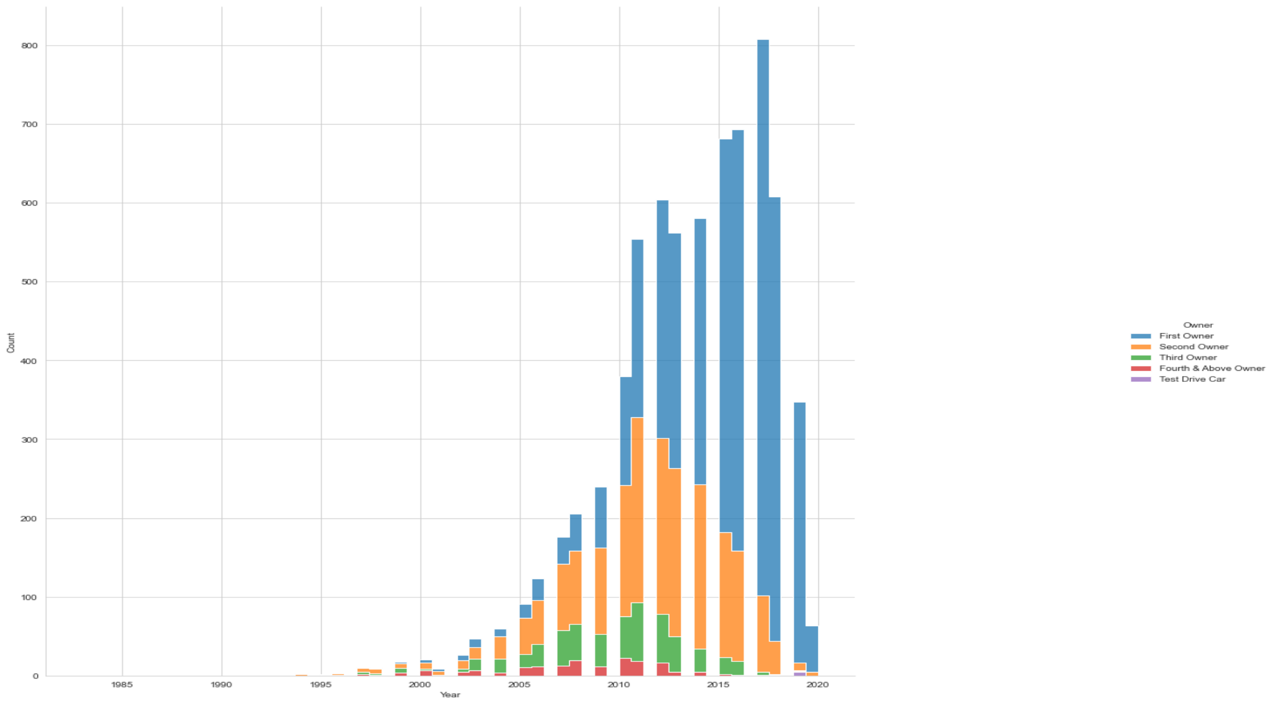
-->The tenth histplot shows that most of the used vehicles have an engine CC of 1000 and least number of vehicles have an engine CC of around 1500. –

->The eleventh histplot shows that most of the used vehicles have a max power around 70bhp while least vehicles have a max power around 100bhp.

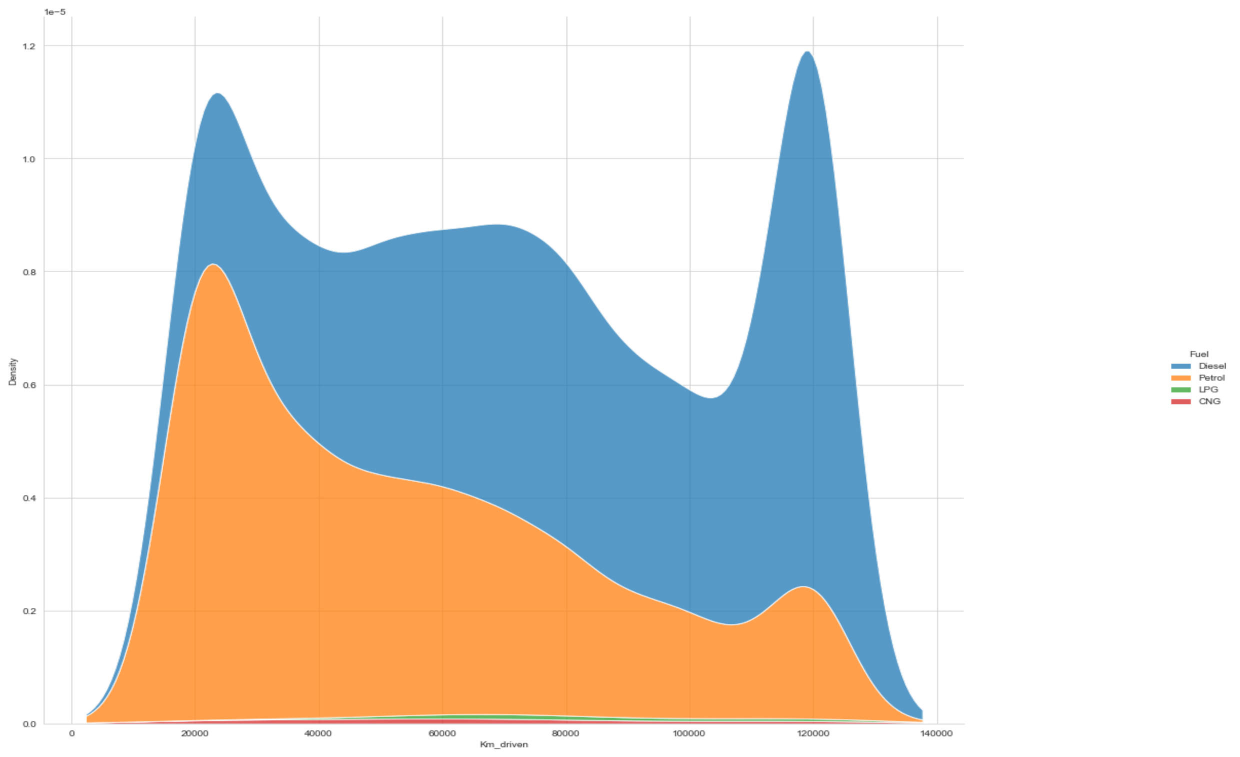
Bivariate Analysis:



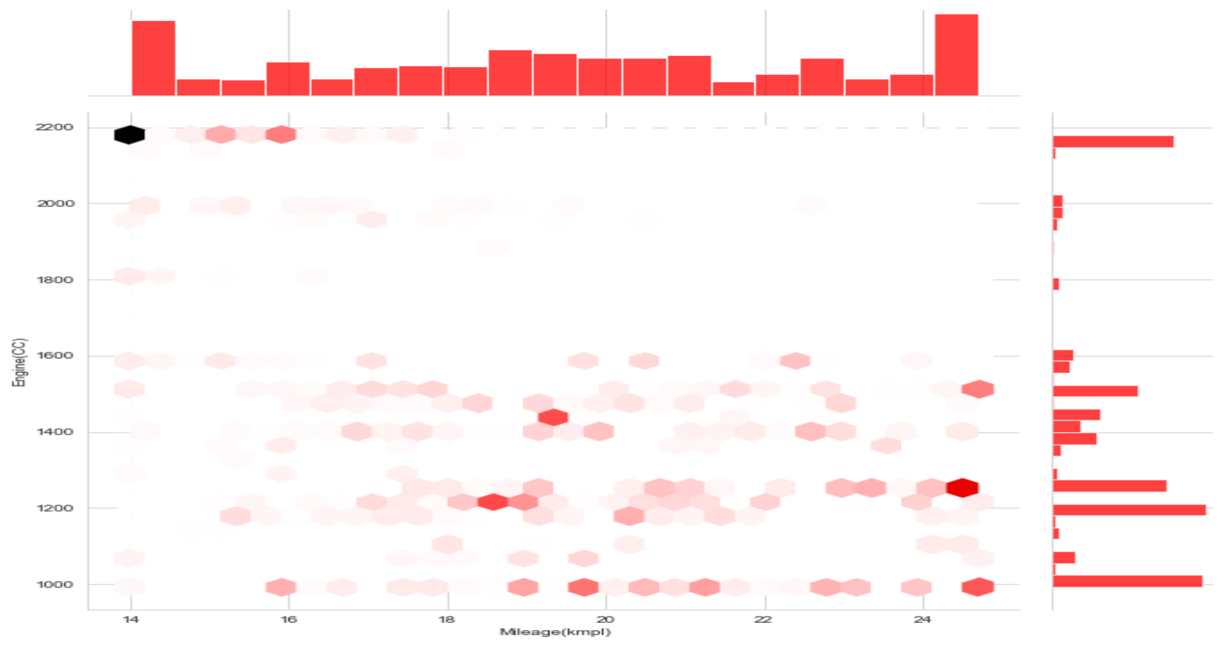
The above line plot shows that vehicles that use diesel have more engine capacity and LPG vehicles have least engine capacity



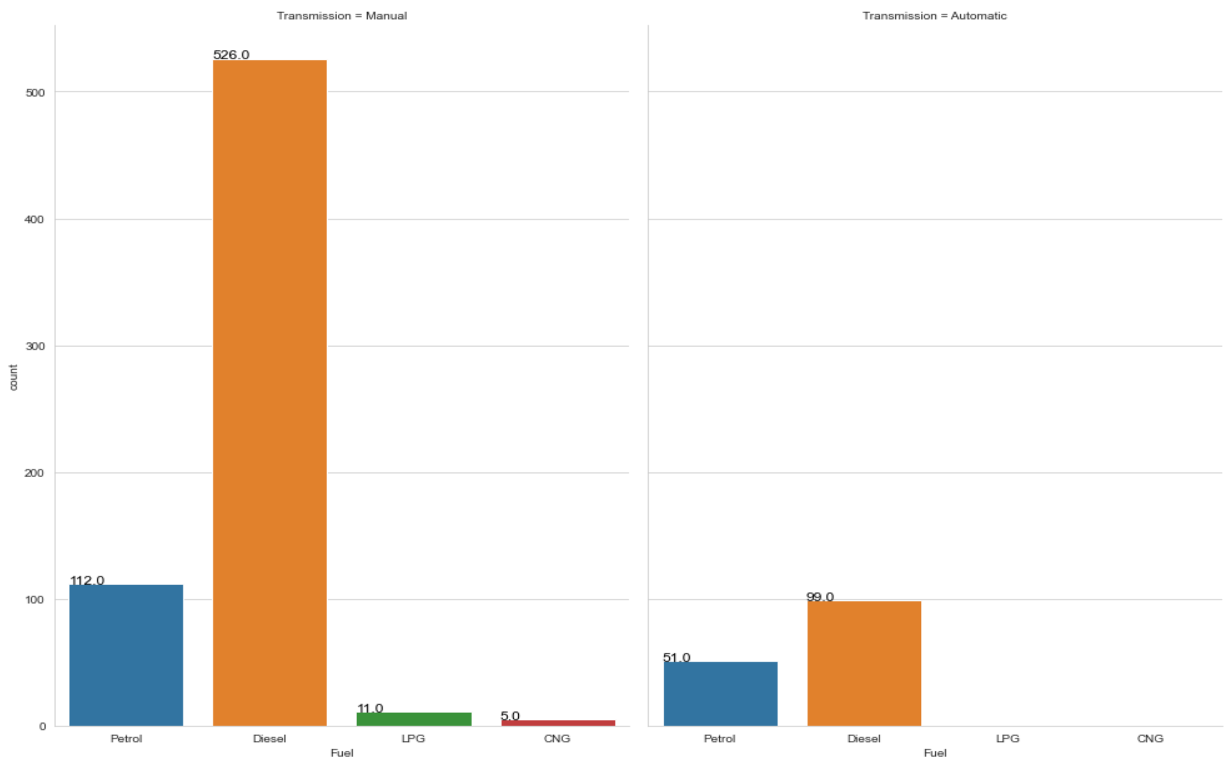
The above histogram shows that in the year 2017 more vehicles were bought and these vehicles belong to first owner.



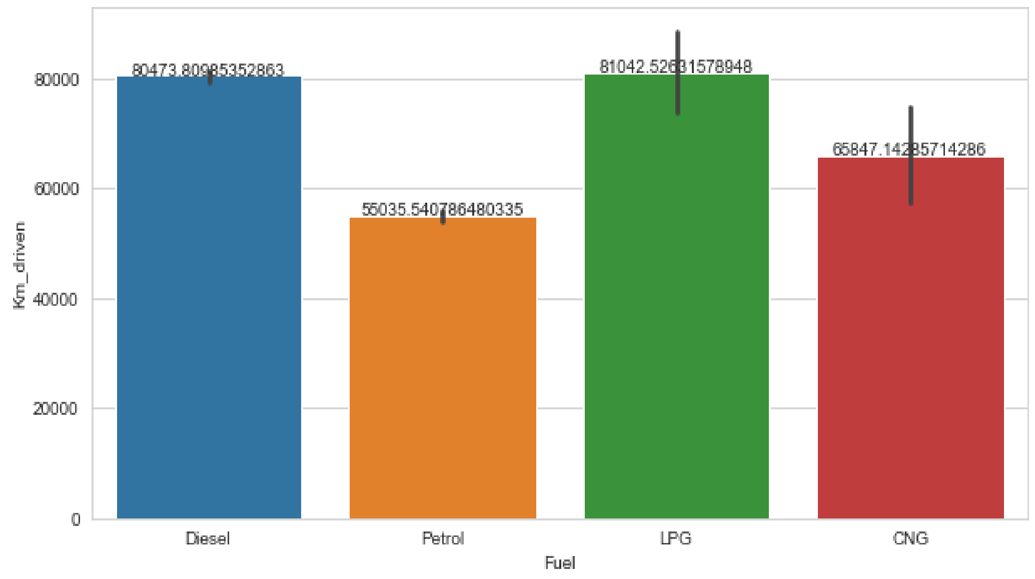
The above kdeplot shows that vehicles that use diesel have driven more kilometer while vehicles that run using CNG have driven the least kilometers.



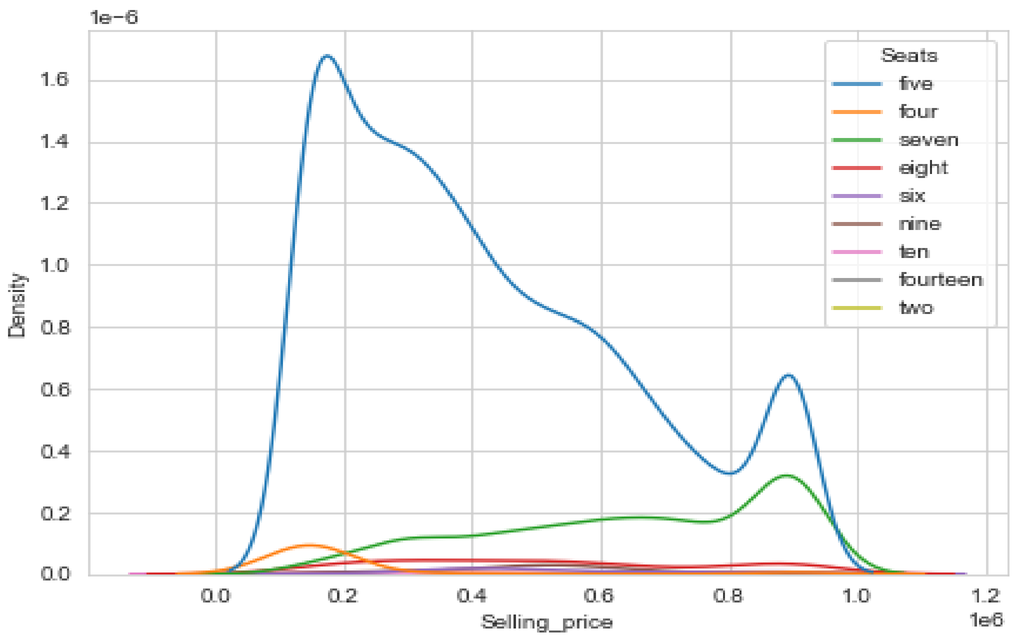
The above jointplot shows that many vehicles with a mileage of 14kmpl has an engine capacity of around 2200CC.



The above countplot shows that most vehicles that use diesel and are manual have mileage less than 14.5kmpl while least vehicles that use CNG and are manual have mileage less than 14.5kmpl

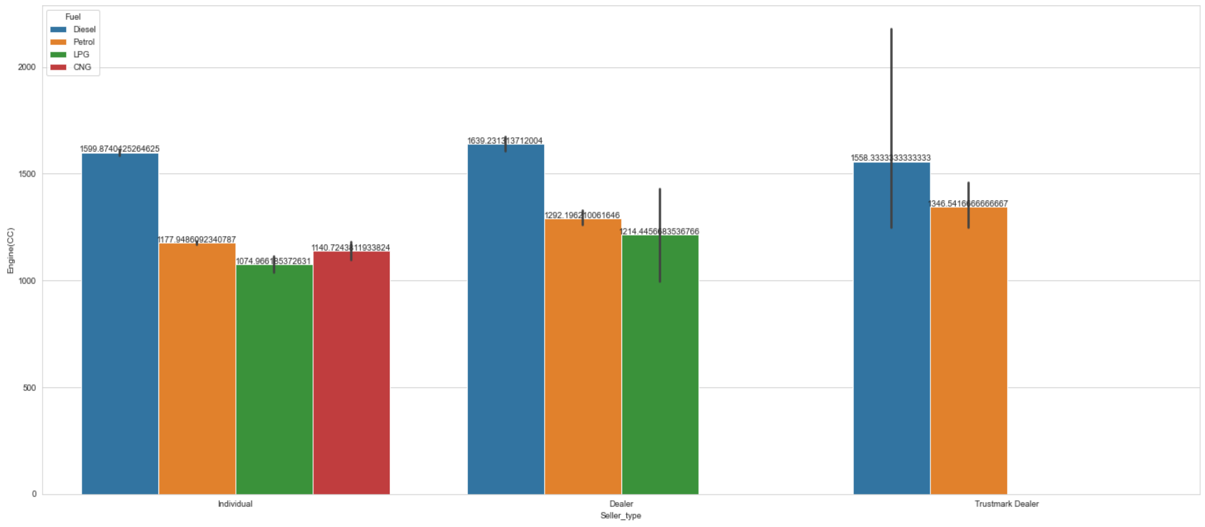


The above barplot shows that vehicles that use LPG have driven more kilometers when compared to vehicles that use petrol, diesel and CNG

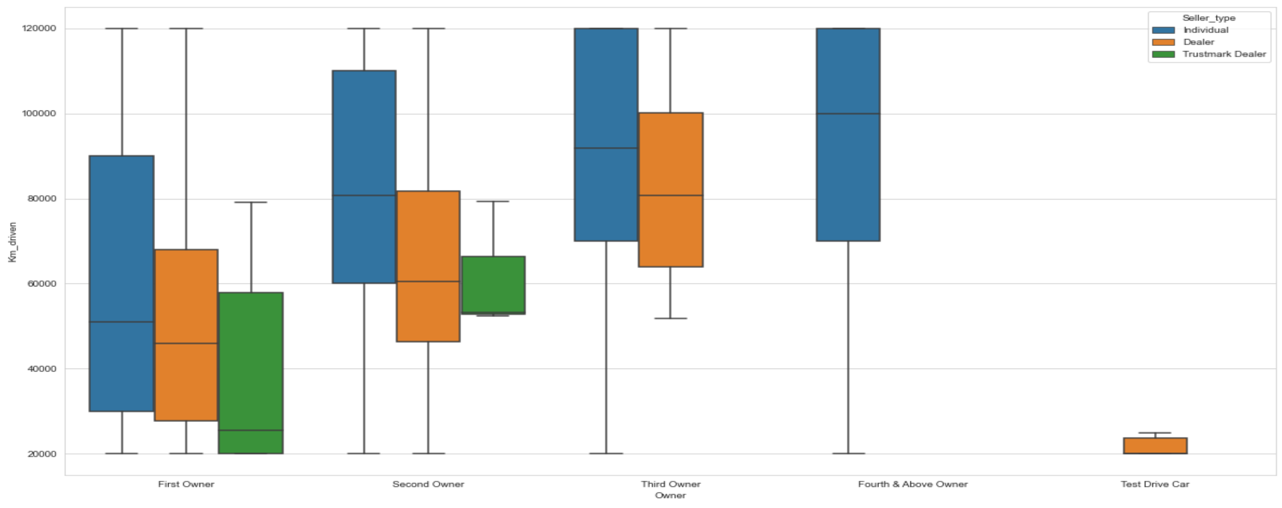


The above kde plot shows that the selling price of five-seater vehicles is more the others

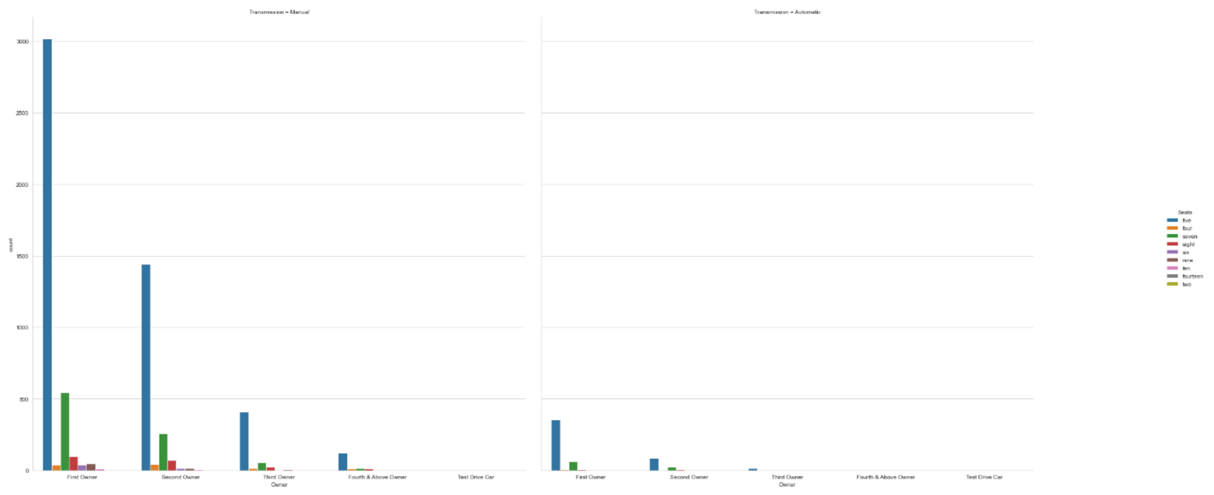
Multivariate Analysis



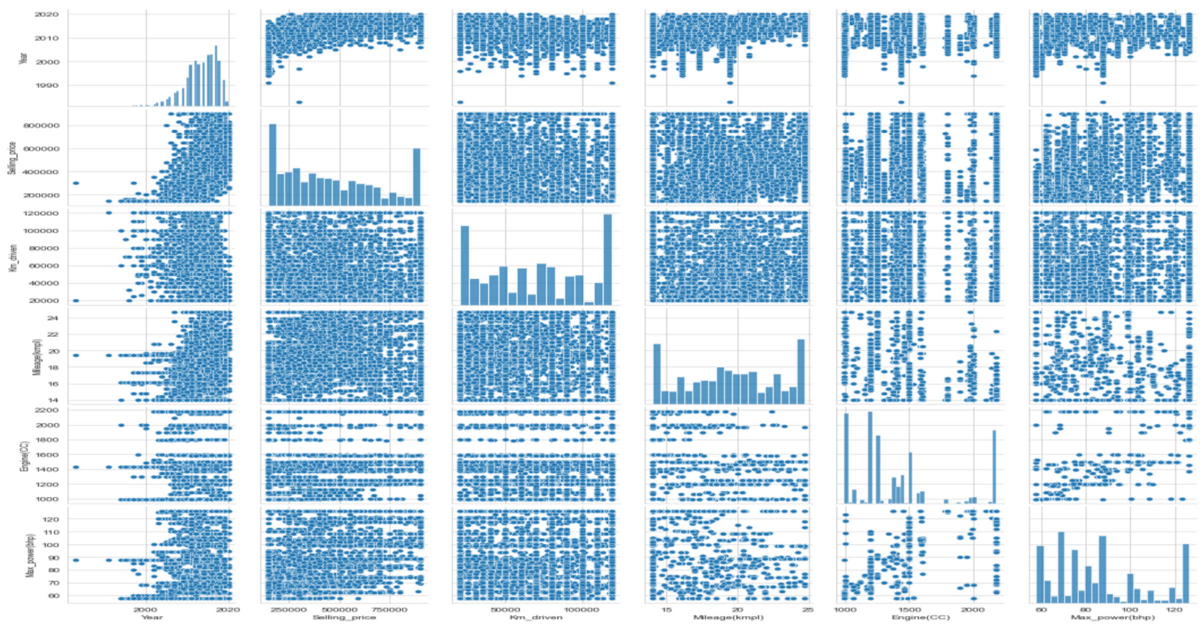
The above kde plot shows that the selling price of five-seater vehicles is more the others



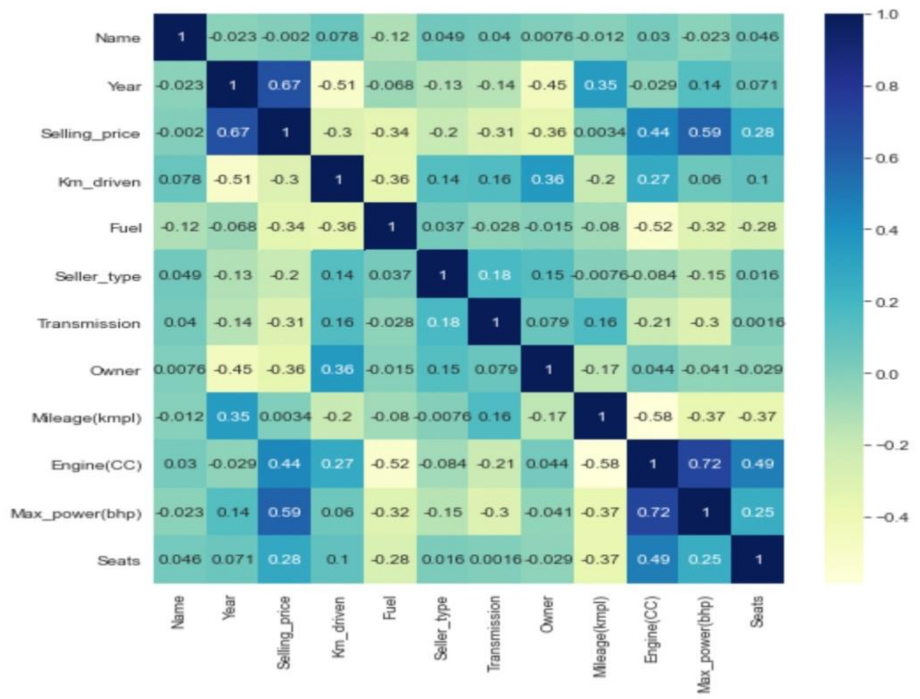
The above kde plot shows that the selling price of five-seater vehicles is more the others



The above countplot shows that five-seater vehicles that are owned by first owners and are manual is more compared to others



Multi Variate pair plot is used to **analyze the pairwise relationship of all the numerical attributes in the dataset.**



This heatmap shows relation between every attribute. So, from this we can say that we may consider correlation if correlation between them is greater than 0.5

MODEL FITTING:

Without PCA

1. Decision tree:

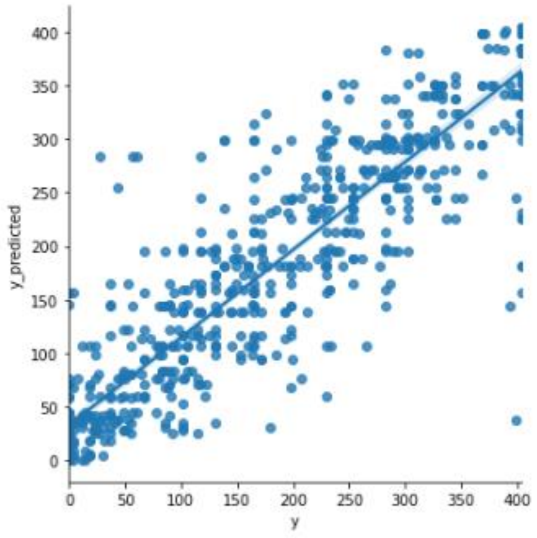
```
decision_tree=DecisionTreeRegressor(min_samples_leaf=.01)
decision_tree.fit(x_train1,y_train1)
y_preds=decision_tree.predict(x_test1)
accuracy.append(r2_score(y_test1,y_preds)*100)
model.append('Decision Tree')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds))
print("Accuracy of Decision Tree Regressor without PCA: ",r2_score(y_test1,y_preds)*100)
print("The mean squared error of Decision tree regressor without pca is: ",mean_squared_error(y_test1,y_preds))
print(" ")
```

```
Accuracy of Decision Tree Regressor without PCA:  81.48515608309188
The mean squared error of Decision tree regressor without pca is:  3235.3791426987646
```

```
df=pd.DataFrame({'y':y_test1,'y_predicted':y_preds})
df
```

	y	y_predicted
5355	404	398.205882
6957	404	381.069767
583	334	270.913043
7413	320	339.712500
5797	165	299.311475
...	...	...
6717	232	233.857143
2784	0	4.623529
5756	404	385.245455
3801	3	0.828571
5487	211	264.726027

693 rows × 2 columns



Accuracy is 81.485%

2. Random Forest:

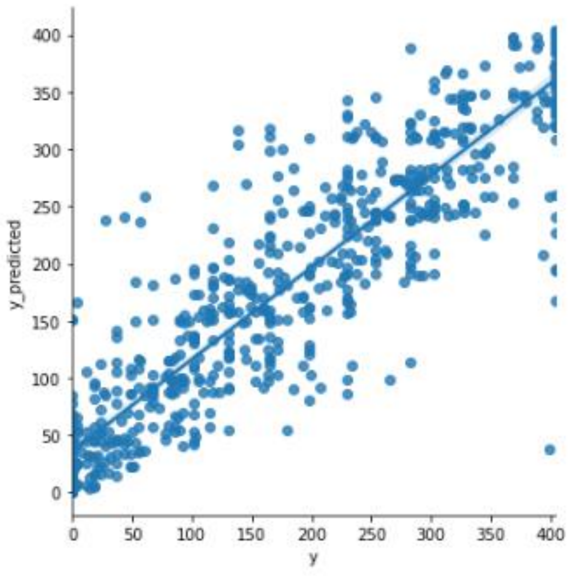
```
randomForest=RandomForestRegressor(min_samples_leaf=.01)
randomForest.fit(x_train1,y_train1)
y_preds=randomForest.predict(x_test1)
accuracy.append(r2_score(y_test1,y_preds)*100)
model.append('randomForest')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds))
print("Accuracy of randomForest Regressor without PCA: ",r2_score(y_test1,y_preds)*100)
print("The mean squared error of randomForest regressor without pca is: ",mean_squared_error(y_test1,y_preds))
print(" ")
```

```
Accuracy of randomForest Regressor without PCA:  81.91380069890215
The mean squared error of randomForest regressor without pca is:  3229.584759728858
```

```
df=pd.DataFrame({'y':y_test1,'y_predicted':y_preds})
df
```

	y	y_predicted
5355	404	397.407447
6957	404	356.054288
583	334	275.049560
7413	320	343.656911
5797	165	310.418504
...	...	...
6717	232	264.678882
2784	0	4.510518
5756	404	373.513678
3801	3	5.518799
5487	211	266.343353

693 rows × 2 columns



Accuracy is 81.9138%

3. Linear Regressor:

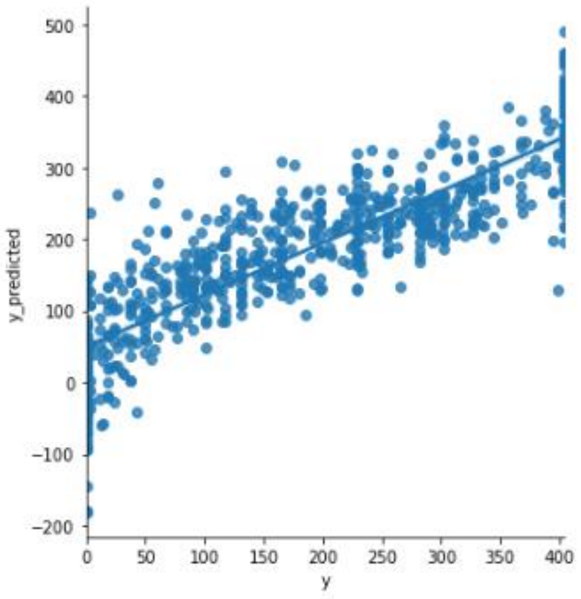
```
mlrm =LinearRegression()
mlrm.fit(x_train1, y_train1)
y_preds1= mlrm.predict(x_test1)
print("the accuracy of linear regression model without PCA is: ",r2_score(y_test1,y_preds1)*100)
print("The mean squared error of Linear regression without pca is: ", mean_squared_error(y_test1,y_preds1))
model.append('Linear regression')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds1))
accuracy.append(r2_score(y_test1,y_preds1)*100)
```

```
the accuracy of linear regression model without PCA is:  72.72573332050465
The mean squared error of Linear regression without pca is:  4766.04577080222
```

```
df1=pd.DataFrame({'y':y_test1,'y_predicted':y_preds1}
df1
```

	y	y_predicted
5355	404	350.055922
6957	404	326.683606
583	334	270.481941
7413	320	273.029501
5797	165	308.218794
...	...	...
6717	232	255.035988
2784	0	-28.359452
5756	404	350.440094
3801	3	-37.450360
5487	211	215.306336

693 rows × 2 columns



Accuracy is 72.725%

4. KNN Regressor:

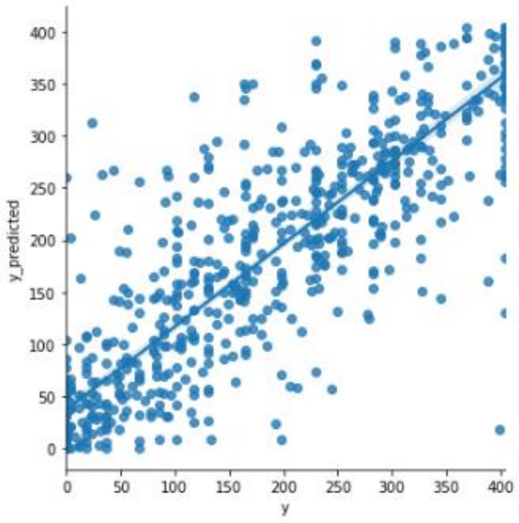
```
neigh= KNeighborsRegressor(n_neighbors=3)
neigh.fit(x_train1, y_train1)
y_preds2= neigh.predict(x_test1)
print("the accuracy of this model is: ",r2_score(y_test1,y_preds2)*100)
print("The mean squared error of KNN regressor without pca is: ", mean_squared_error(y_test1,y_preds2))
model.append('KNN regressor')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds2))
accuracy.append(r2_score(y_test1,y_preds2)*100)
```

```
the accuracy of this model is:  75.49314191838229
The mean squared error of KNN regressor without pca is:  4282.454545454545
```

```
df2=pd.DataFrame({'y':y_test1,'y_predicted':y_preds2})
df2
```

	y	y_predicted
5355	404	404.000000
6957	404	390.666667
583	334	286.666667
7413	320	294.666667
5797	165	251.666667
...	...	...
6717	232	181.333333
2784	0	0.000000
5756	404	378.000000
3801	3	39.000000
5487	211	237.666667

693 rows × 2 columns



Accuracy is 75.4931%

Hyper parameter tuning:

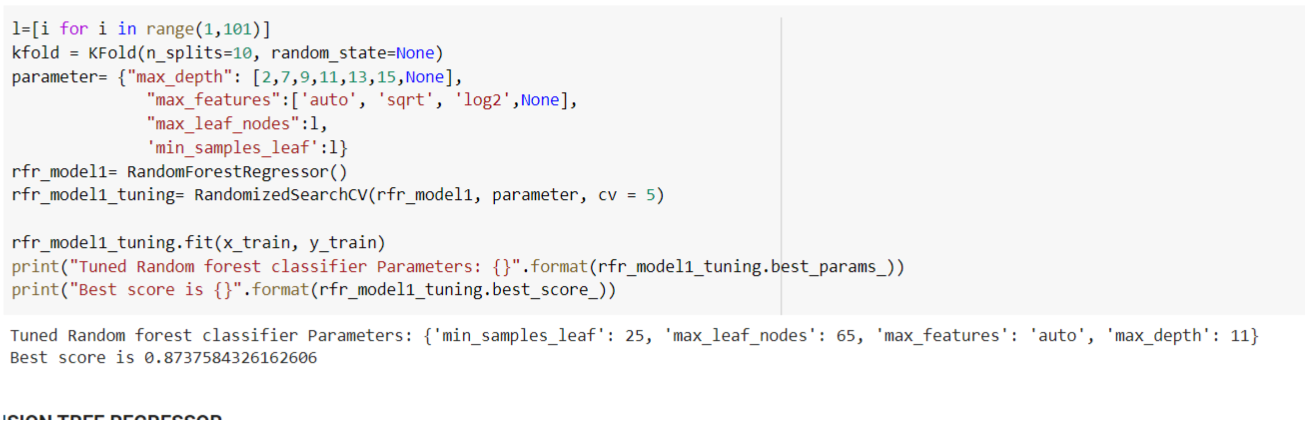
Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. As ML Models are not intelligent enough to know what hyperparameters would lead to the highest possible accuracy on the given dataset so we allow the models to try different combinations of hyperparameter during the training and testing process in order to get the best result.

Some of the hyperparameters

- In Random Forest are:
    - n_estimators (total number of trees in a forest)
    - max_depth (the depth of each tree in the forest)
    - criterion (the method to make splits in each tree).
- In Decision Tree are:
    - min_samples_split (minimum number of samples required to split an internal node)
    - max_depth (the depth of each tree in the forest)
    - min_samples_leaf (minimum number of samples required to be at a leaf node)
    - max_features (the number of features to consider when looking for the best split)
- In KNN are:
    - k (Number of neighbours)

---

1. Random Forest

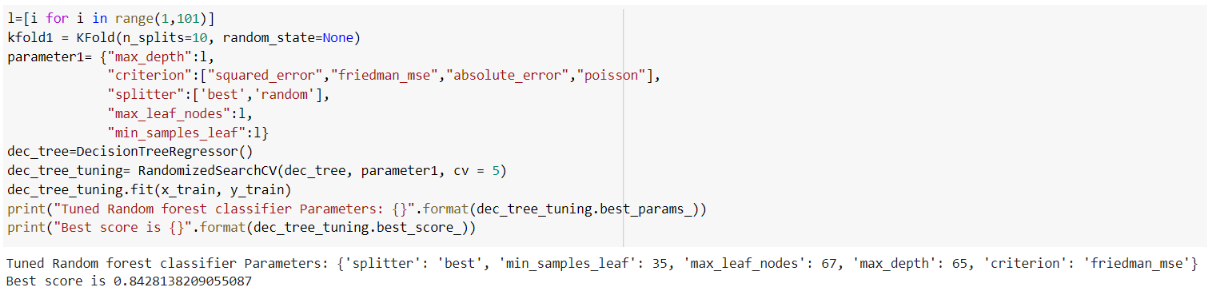


```
l=[i for i in range(1,101)]
kfold = KFold(n_splits=10, random_state=None)
parameter= {"max_depth": [2,7,9,11,13,15,None],
                "max_features":['auto', 'sqrt', 'log2',None],
                "max_leaf_nodes":l,
                'min_samples_leaf':l}
rfr_model1= RandomForestRegressor()
rfr_model1_tuning= RandomizedSearchCV(rfr_model1, parameter, cv = 5)

rfr_model1_tuning.fit(x_train, y_train)
print("Tuned Random forest classifier Parameters: {}".format(rfr_model1_tuning.best_params_))
print("Best score is {}".format(rfr_model1_tuning.best_score_))
```

```
Tuned Random forest classifier Parameters: {'min_samples_leaf': 25, 'max_leaf_nodes': 65, 'max_features': 'auto', 'max_depth': 11}
Best score is 0.8737584326162606
```

ISION TREE REGRESSOR

2. Decision Trees



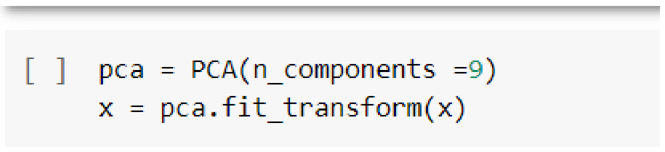
```
l=[i for i in range(1,101)]
kfold1 = KFold(n_splits=10, random_state=None)
parameter1= {"max_depth":l,
             "criterion":["squared_error","friedman_mse","absolute_error","poisson"],
             "splitter":['best','random'],
             "max_leaf_nodes":l,
             "min_samples_leaf":l}
dec_tree=DecisionTreeRegressor()
dec_tree_tuning= RandomizedSearchCV(dec_tree, parameter1, cv = 5)
dec_tree_tuning.fit(x_train, y_train)
print("Tuned Random forest classifier Parameters: {}".format(dec_tree_tuning.best_params_))
print("Best score is {}".format(dec_tree_tuning.best_score_))
```

```
Tuned Random forest classifier Parameters: {'splitter': 'best', 'min_samples_leaf': 35, 'max_leaf_nodes': 67, 'max_depth': 65, 'criterion': 'friedman_mse'}
Best score is 0.8428138209055087
```

Optimal parameters for Decision Trees.

Principal Component Analysis (PCA):

This is mainly for dimensionality reduction for data frame. So, that only important attributes can be part of the model which in turn helps us to improve the accuracy of model.



```
[ ]  pca = PCA(n_components =9)
     x = pca.fit_transform(x)
```

PCA + Bagging:

Bagging is an ensemble learning method, used to reduce variance of model and to increase accuracy of model.

In bagging, a random sample of data in a training set is selected with replacement.

After generating several data samples, these weak models are then trained independently, and depending on the type of task regression or classification, for example the average or majority of those predictions yield a more accurate estimate.

1. Decision Tree:

```
x_train2, x_test2, y_train2, y_test2 = train_test_split(x,y,test_size=0.10)
```

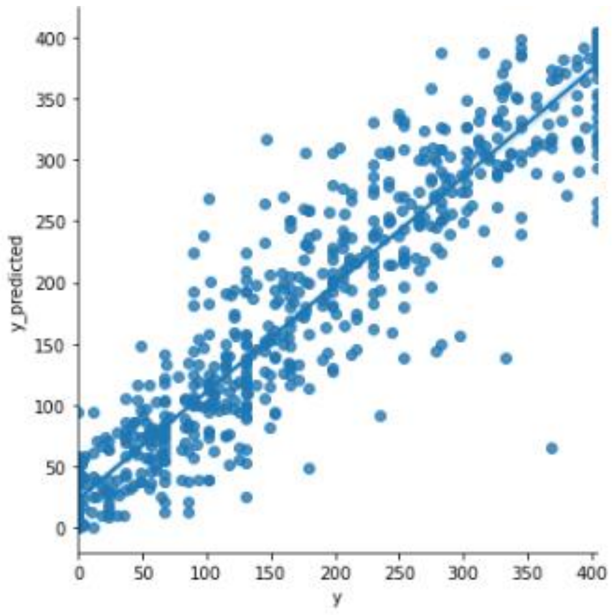
```
decision_tree1=DecisionTreeRegressor()
num=90
model1= BaggingRegressor(base_estimator=decision_tree1, n_estimators=num)
model1.fit(x_train2,y_train2)
y_preds3=model1.predict(x_test2)
print('Accuracy of Decision Tree Classifier is ',r2_score(y_test2,y_preds3)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds3))
mse.append(mean_squared_error(y_test2,y_preds3))
model.append('Decision Tree')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds3)*100)
```

```
Accuracy of Decision Tree Classifier is  87.84184381926805
the mean squared error of the decision tree classifier with PCA and bagging is:  2083.1618927280756
```

```
df3=pd.DataFrame({'y':y_test2,'y_predicted':y_preds3})
df3
```

	y	y_predicted
6455	404	376.638889
3507	326	286.770370
7057	24	15.750000
5518	173	173.411111
2262	55	71.411111
...	...	...
4914	207	255.655556
445	115	96.514815
276	168	184.107407
2177	198	279.372222
709	0	4.847222

693 rows × 2 columns



2. Linear Regression

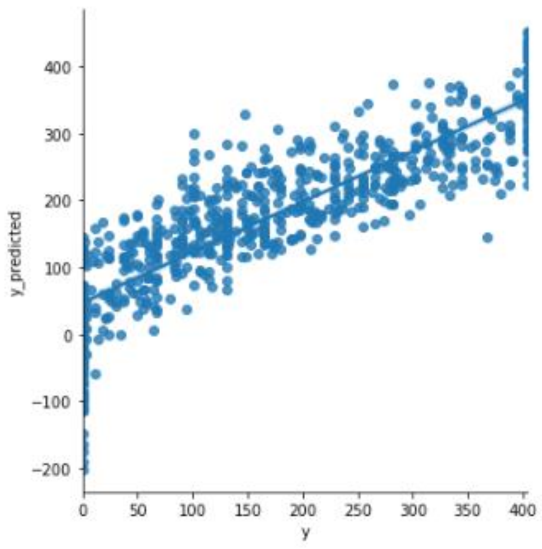
```
mlrm1=LinearRegression()
num1=90
model2= BaggingRegressor(base_estimator=mlrm1, n_estimators=num1)
model2.fit(x_train2, y_train2)
y_preds4= model2.predict(x_test2)
print('Accuracy of linear regression method is ',r2_score(y_test2,y_preds4)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds4))
mse.append(mean_squared_error(y_test2,y_preds4))
model.append('Linear regression')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds4)*100)
```

```
Accuracy of linear regression method is  73.79916710856193
the mean squared error of the decision tree classifier with PCA and bagging is:  4489.2149620251275
```

```
df4=pd.DataFrame({'y':y_test2,'y_predicted':y_preds4})
df4
```

	y	y_predicted
6455	404	370.197001
3507	326	219.325262
7057	24	113.981437
5518	173	136.380400
2262	55	152.368163
...	...	...
4914	207	182.590112
445	115	120.984467
276	168	264.405521
2177	198	275.853430
709	0	48.218584

693 rows × 2 columns



3. KNN neighbour

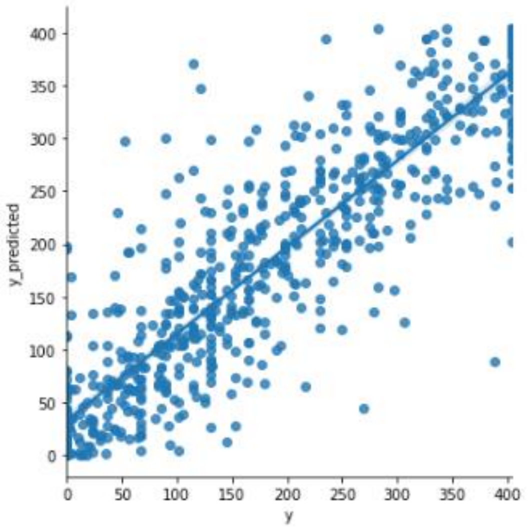
```
neigh1= KNeighborsRegressor(n_neighbors=3)
num2=90
model3= BaggingRegressor(base_estimator=neigh1, n_estimators=num2)
model3.fit(x_train2, y_train2)
y_preds5= model3.predict(x_test2)
print("the accuracy of this model is: ",r2_score(y_test2,y_preds5)*100)
print("The mean squared error of KNN regressor without pca is: ", mean_squared_error(y_test2,y_preds5))
model.append('KNN regressor')
with_pca.append(1)
mse.append(mean_squared_error(y_test2,y_preds5))
accuracy.append(r2_score(y_test2,y_preds5)*100)
```

```
the accuracy of this model is:  79.40135074591821
The mean squared error of KNN regressor without pca is:  3529.344460616354
```

```
df5=pd.DataFrame({'y':y_test2,'y_predicted':y_preds5})
df5
```

	y	y_predicted
6455	404	358.074074
3507	326	292.296296
7057	24	76.188889
5518	173	160.540741
2262	55	63.077778
...	...	...
4914	207	230.188889
445	115	161.481481
276	168	206.618519
2177	198	180.174074
709	0	3.570370

693 rows × 2 columns

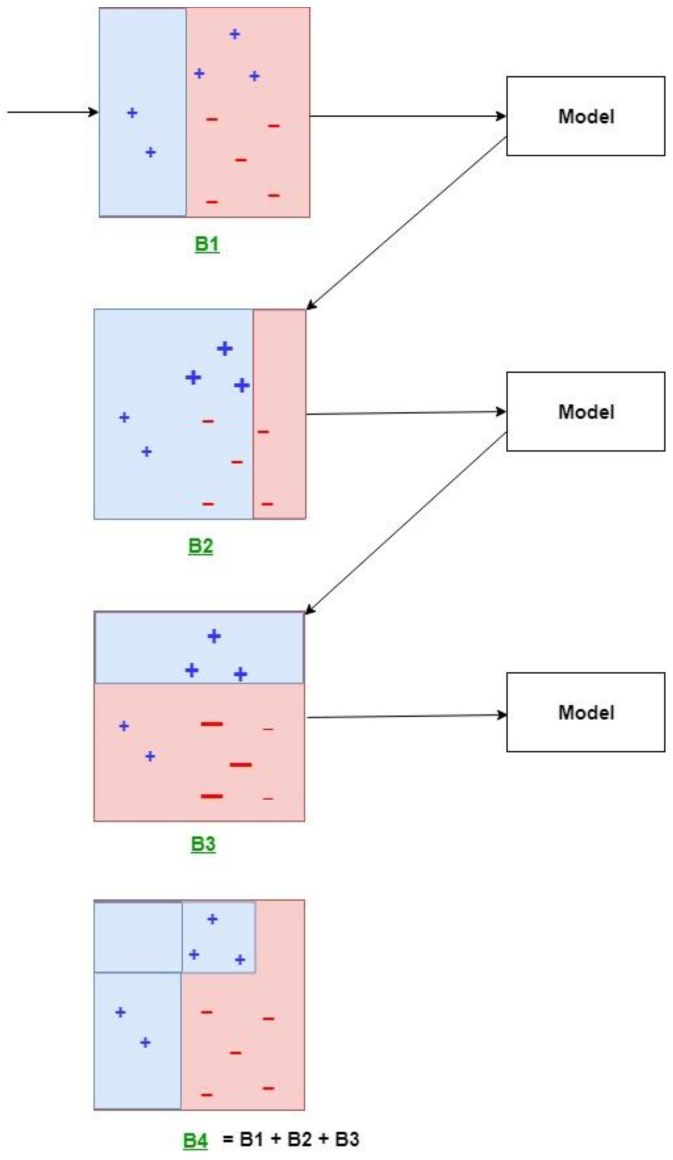


Boosting:

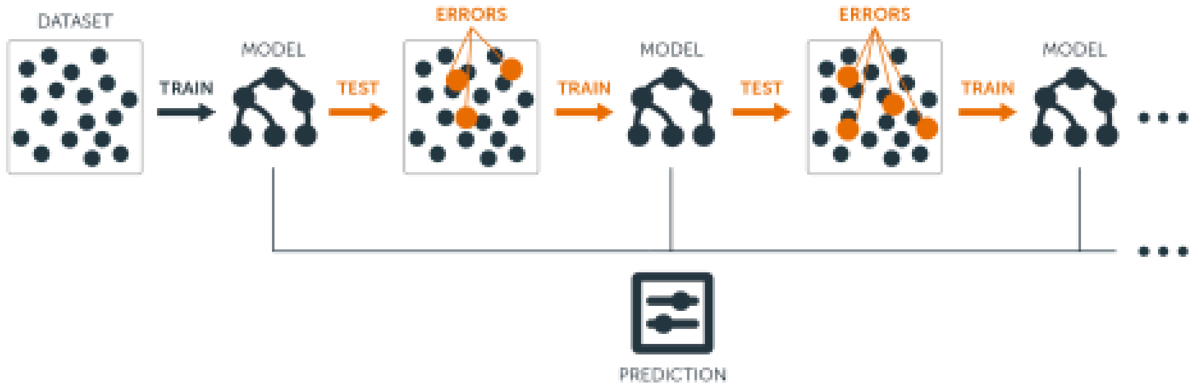
Boosting is an ensemble modelling technique that attempts to build a strong classifier from the number of weak classifiers.

To find weak rule, we apply ML algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Ada Boosting:

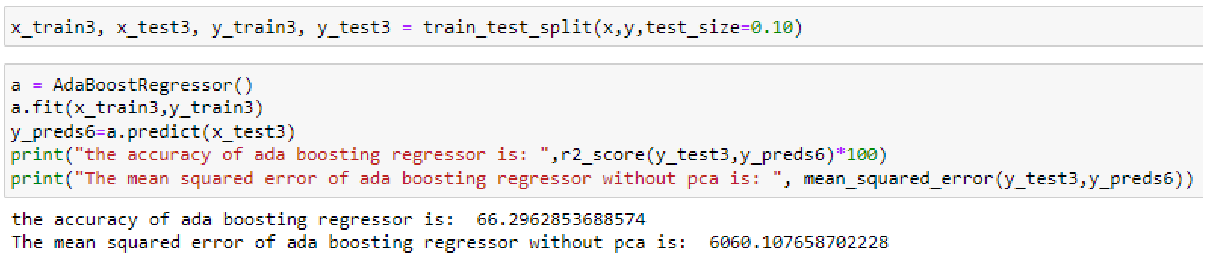


Gradient Boosting



---

1. Ada Boosting:



```
x_train3, x_test3, y_train3, y_test3 = train_test_split(x,y,test_size=0.10)
```

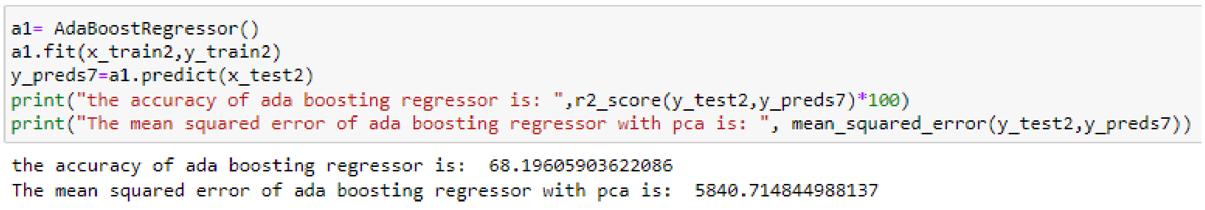
```
a = AdaBoostRegressor()
a.fit(x_train3,y_train3)
y_preds6=a.predict(x_test3)
print("the accuracy of ada boosting regressor is: ",r2_score(y_test3,y_preds6)*100)
print("The mean squared error of ada boosting regressor without pca is: ", mean_squared_error(y_test3,y_preds6))
```

```
the accuracy of ada boosting regressor is:  66.2962853688574
The mean squared error of ada boosting regressor without pca is:  6060.107658702228
```

Accuracy is 66.296%

---

2. Ada Boosting + PCA:



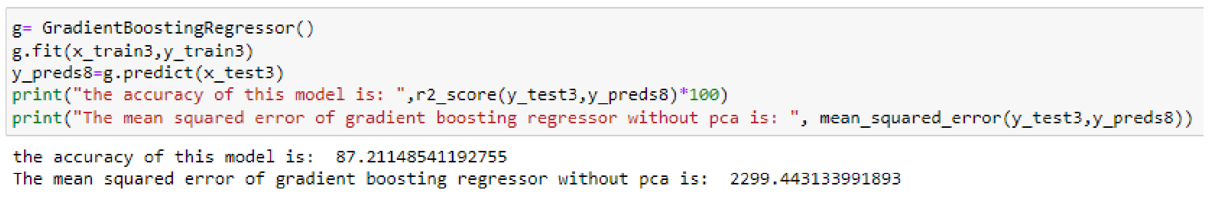
```
a1= AdaBoostRegressor()
a1.fit(x_train2,y_train2)
y_preds7=a1.predict(x_test2)
print("the accuracy of ada boosting regressor is: ",r2_score(y_test2,y_preds7)*100)
print("The mean squared error of ada boosting regressor with pca is: ", mean_squared_error(y_test2,y_preds7))
```

```
the accuracy of ada boosting regressor is:  68.19605903622086
The mean squared error of ada boosting regressor with pca is:  5840.714844988137
```

Accuracy is 68.196%

---

3. Gradient Boosting:

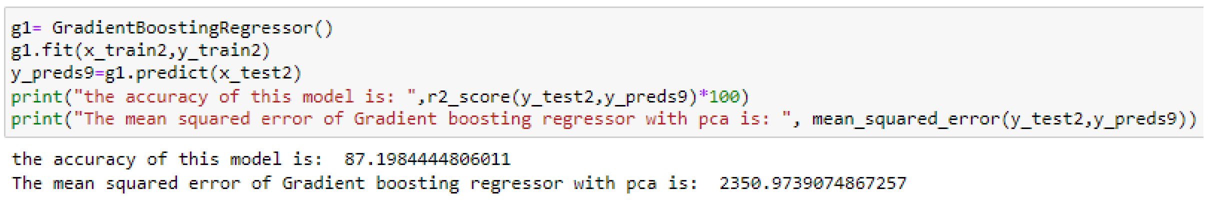


```
g= GradientBoostingRegressor()
g.fit(x_train3,y_train3)
y_preds8=g.predict(x_test3)
print("the accuracy of this model is: ",r2_score(y_test3,y_preds8)*100)
print("The mean squared error of gradient boosting regressor without pca is: ", mean_squared_error(y_test3,y_preds8))
```

```
the accuracy of this model is:  87.21148541192755
The mean squared error of gradient boosting regressor without pca is:  2299.443133991893
```

Accuracy is 87.211%

4. Gradient Boosting + PCA:



```
g1= GradientBoostingRegressor()
g1.fit(x_train2,y_train2)
y_preds9=g1.predict(x_test2)
print("the accuracy of this model is: ",r2_score(y_test2,y_preds9)*100)
print("The mean squared error of Gradient boosting regressor with pca is: ", mean_squared_error(y_test2,y_preds9))
```

```
the accuracy of this model is:  87.1984444806011
The mean squared error of Gradient boosting regressor with pca is:  2350.9739074867257
```

Accuracy is 87.198%

Comparison in Boosting:

	without PCA	with PCA
Ada Boosting	66.296%	68.196%
Gradient Boosting	87.211%	87.198%

Conclusion:

Our main aim to work on this project is to find cars with best qualities(attributes) in affordable manner i.e., to find best cars at suitable prices. It is easier to get but we need to increase accuracy of model so that we get best results. So, Hyper parameter tuning, PCA, Bagging and Boosting are used to increase the accuracy of the model.