FLIP ROBO

Used car price prediction

Submitted by:

Allen Paul

ACKNOWLEDGMENT

INTRODUCTION

• Business Problem Framing

The prices of new cars in the industry is 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 (Pal, Arora and Palakurthy, 2018). 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

The Client: To be able to predict used cars market value can help both buyers and sellers.

Used car sellers (dealers): They are one of the biggest target group that can be interested in results of this study. If used car sellers better understand what makes a car desirable, what the important features are for a used car, then they may consider this knowledge and offer a better service.

Online pricing services: There are websites that offers an estimate value of a car. They may have a good prediction model. However, having a second model may help them to give a better prediction to their users. Therefore, the model developed in this study may help online web services that tells a used car's market value.

Individuals: There are lots of individuals who are interested in the used car market at some points in their life because they wanted to sell their car or buy a used car. In this process, it's a big corner to pay too much or sell less then it's market value

• Conceptual Background of the Domain Problem

Problem statement:

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. Therefore we need a ML model in order to predict the price of the used car based on features such as Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location etc.

- ## Review of Literature

  After researching about the used car price prediction problem, we can see that we need to keep updating our model by working on with fresh data from time to time to get the impact of the condition during that period so that we can predict more accurately.

- ## Motivation for the Problem Undertaken

  The motivation for undertaking this problem is to make a model keeping all the features effecting the price and using them to train the model which helps us to predict used car price more accurately

Analytical Problem Framing

- ## Mathematical/ Analytical Modeling of the Problem

  We have collected the data from olx.com of different cars from different states in India by scraping them using selenium.

  The null value rows were removed in order to have a better data set as having null values will affect in training the model.

  The Sample of the dataset is shown below:

```
df = pd.read_csv("dfcar.csv")
#importing the file which we have extracted using selenium
pd.set_option('display.max_columns', None) # displays maximum columns
df
```

	Brand	Model	Variant	Manufacturig year	Kilometers driven	Fuel	No of Owners	Location	Price
0	Toyota	Innova Crysta	2.4 V	2017	53,000 km	Diesel	2nd	Puthanathani, Malappuram, Kerala	₹ 18,50,000
1	BMW	5 Series	520d Luxury Line	2011	140,000 km	Diesel	1st	Koduvally, Kozhikode, Kerala	₹ 10,45,000
2	Renault	KWID	Climber 1.0 MT Opt	2018	13,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,70,000
3	Renault	KWID	1.0 RXT AMT Opt	2017	23,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,80,000
4	Ford	Ikon	1.4 TDCi DuraTorq	2010	99,000 km	Diesel	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 1,09,000
5	Hyundai	i10	1.2 Kappa Magna	2013	36,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 2,85,000
6	Maruti Suzuki	Celerio	VXI AMT	2014	51,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,80,000

Data Sources and their formats

The data has been scraped from olx.com from various states in India.

In this dataset we have the following columns:

"Brand", **"Model"**, **"Variant"**, **"Manufacturig year"**, **"Kilometers driven"**, **"Fuel"**, **"No of Owners"**, **"Location"** and **"Price"**

Data types of the columns are as shown below:

```
df.dtypes #checking data types

Brand                 object
Model                 object
Variant               object
Manufacturig year      int64
Kilometers driven     object
Fuel                  object
No of Owners          object
Price                 object
State                 object
dtype: object
```

Data Preprocessing Done

Importing the data:

```
df = pd.read_csv("dfcar.csv")
#importing the file which we have extracted using selenium
pd.set_option('display.max_columns', None) # displays maximum columns
df
```

	Brand	Model	Variant	Manufacturig year	Kilometers driven	Fuel	No of Owners	Location	Price
0	Toyota	Innova Crysta	2.4 V	2017	53,000 km	Diesel	2nd	Puthanathani, Malappuram, Kerala	₹ 18,50,000
1	BMW	5 Series	520d Luxury Line	2011	140,000 km	Diesel	1st	Koduvally, Kozhikode, Kerala	₹ 10,45,000
2	Renault	KWID	Climber 1.0 MT Opt	2018	13,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,70,000
3	Renault	KWID	1.0 RXT AMT Opt	2017	23,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,80,000
4	Ford	Ikon	1.4 TDCi DuraTorq	2010	99,000 km	Diesel	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 1,09,000
5	Hyundai	i10	1.2 Kappa Magna	2013	36,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 2,85,000
6	Maruti Suzuki	Celerio	VXI AMT	2014	51,000 km	Petrol	1st	Kazhakoottam, Thiruvananthapuram, Kerala	₹ 3,80,000

Checking for null or '-' values:

```
: print((df == '-').sum()):  df.isnull().sum() #checking null values.
```

```
Brand                0  :  Brand                0
Model                0     Model                0
Variant              0     Variant              0
Manufacturig year    0     Manufacturig year    0
Kilometers driven    0     Kilometers driven    0
Fuel                 0     Fuel                 0
No of Owners         0     No of Owners         0
Location             0     Location             0
Price                0     Price                0
dtype: int64              dtype: int64
```

There are no null values.

We can see from below that there are lot of unique values in the location column. Therefore, to simplify it we can only take out the states instead of the whole address.

```
df['Location'].nunique()
```

```
1191
```

```
# new data frame with split value columns
df["State"] = df["Location"].str.split(",", expand = True)[2]
```

```
df.drop('Location',axis=1,inplace=True)
```

We have now created a new column state and dropped location

```
df.dtypes #checking data types
```

```
Brand                object
Model                object
Variant              object
Manufacturig year     int64
Kilometers driven    object
Fuel                 object
No of Owners         object
Price                object
State                object
dtype: object
```

We can see that kilometers driver and price is object type. We need to convert them into integers

Removing unnecessary things from the column values

```
df['Kilometers driven']=df['Kilometers driven'].apply(lambda x: x.replace(',',''))
```

```
df['Kilometers driven'] = df['Kilometers driven'].str.extract(r'(\d+[.\d]*)')
```

```
df['Price']=df['Price'].apply(lambda x: x.replace(',',''))
```

```
df['Price'] = df['Price'].str.extract(r'(\d+[.\d]*)')
```

```
df["Kilometers driven"] = pd.to_numeric(df["Kilometers driven"])
```
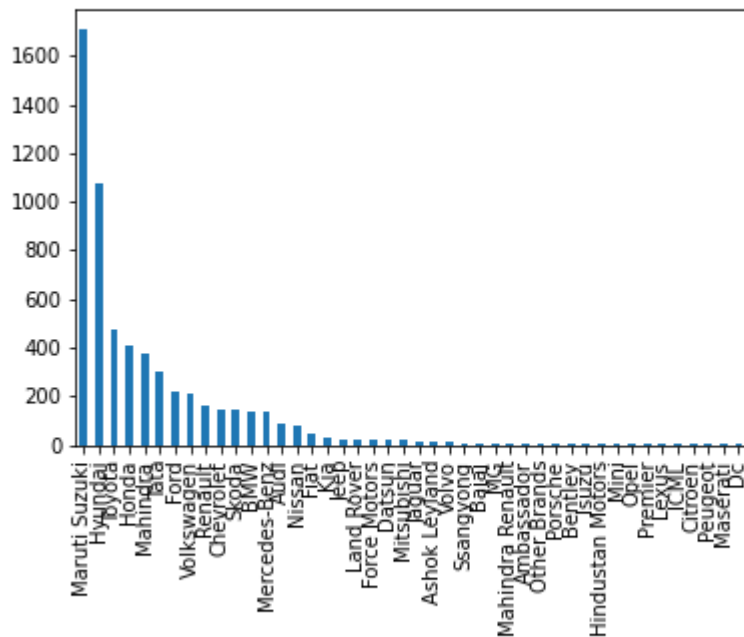
```
df["Price"] = pd.to_numeric(df["Price"])
```

```
df.dtypes
```

```
Brand                object
Model                object
Variant              object
Manufacturig year     int64
Kilometers driven     int64
Fuel                 object
No of Owners         object
Price                 int64
State                object
dtype: object
```

We have now converted kilometers driver and price into integers

```
: df['Brand'].value_counts().plot.bar()
  plt.show()
```


We can see here that maximum of the people buy maruti suzuki brand cars

```
df['Model'].value_counts()
```

```
Swift               288
Swift Dzire         183
Innova              179
i10                 167
i20                 164
Wagon R             152
City                151
Polo                122
XUV500              119
Verna               111
Santro Xing         105
Others               97
Grand i10            96
Creta                94
Alto                 90
Ertiga               84
Baleno               84
Ecosport             84
800                  77
```

we can see that maximum people buy maruti suzuki swift car.

```
df['Variant'].value_counts()
```

```
Others                340
VXI                   138
VDI                   114
LXI                   105
LXi                    47
GLS                    41
Magna                  33
2.5 V 7 STR            31
2011-2014 VDI          30
1.2 Kappa Magna        30
AC                     29
1.6 CRDi SX            29
VDi                    27
1.5 TDI Highline       27
LDI                    26
1.2 Spotz              24
Era                    23
LX                     23
Sportz                 22
```

Here others is highest which we cannot say and therefore is ambigous.

```
df['Variant'].nunique() #checking unique values
```

1924

We can see that there are many unique values in "Variant" Column, and also from the above point we can just drop this column as it won't be helpful.
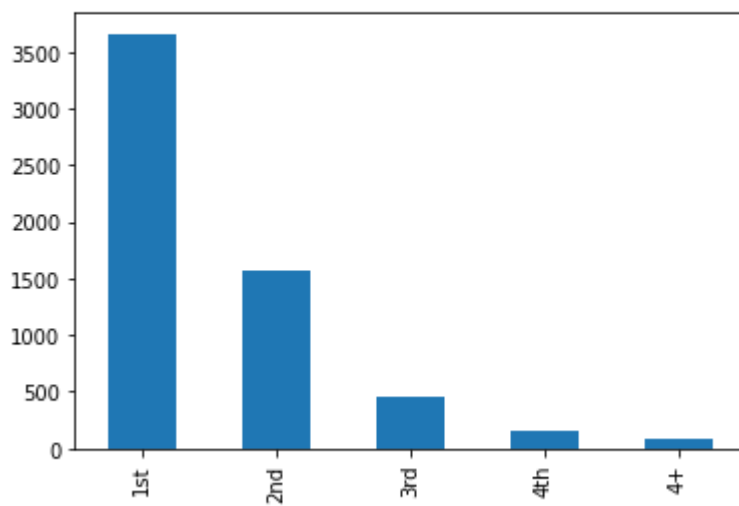
```
df.drop('Variant',axis=1,inplace=True)
```

```
: df['Fuel'].value_counts().plot.bar()
  plt.show()
```
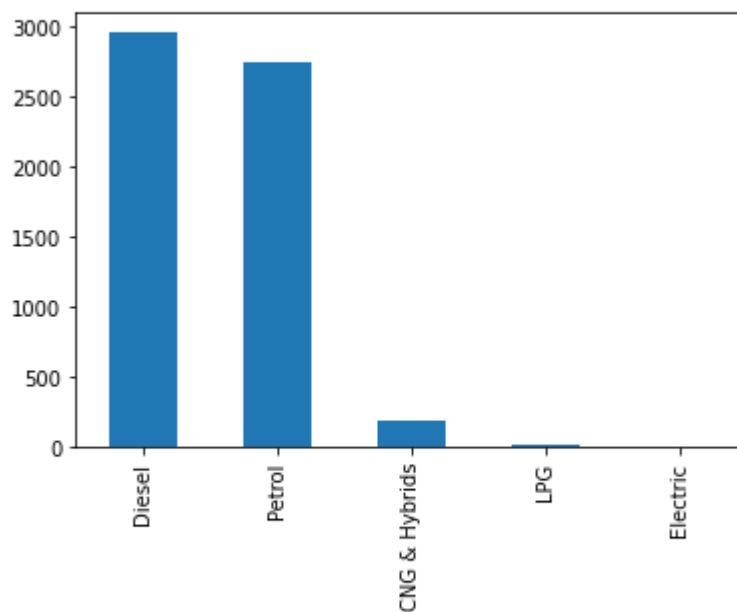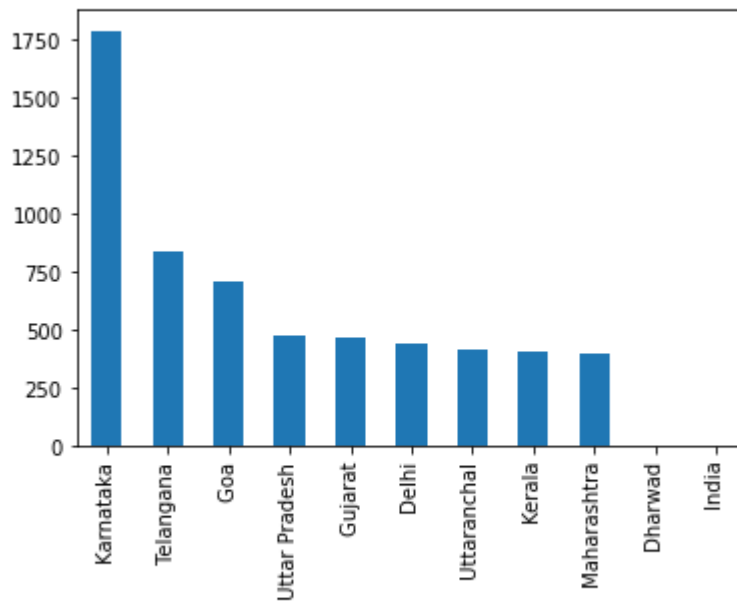

We can see that maximum people prefer to buy diesal cars.

```
df['No of Owners'].value_counts().plot.bar()
plt.show()
```


We can see that maximum people buy car which was owned by only 1 person. Naturally this is because the lesser the owners owned before, better the condition of the car.

```
df['State'].value_counts().plot.bar()
plt.show()
```


We can see that maximum of the used cars are sold in Karnataka

Data Preprocessing

```
#Scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

for i in ['Kilometers driven']:
    df[[i]]=scaler.fit_transform(df[[i]])
```

```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
for i in ['Brand','Model','Manufacturig year','Kilometers driven','Fuel','No of Owners','Price','State']:
    df[i]=label.fit_transform(df[i])
```

```
: df.corr()
```

	Brand	Model	Manufacturig year	Kilometers driven	Fuel	No of Owners	Price	State
Brand	1.000000	0.105976	0.030972	0.125512	-0.111909	0.018655	-0.005285	0.008898
Model	0.105976	1.000000	-0.012140	0.079423	-0.049528	0.000656	-0.092153	0.030636
Manufacturig year	0.030972	-0.012140	1.000000	-0.409764	-0.132295	-0.483513	0.601707	0.018801
Kilometers driven	0.125512	0.079423	-0.409764	1.000000	-0.298181	0.312665	-0.219021	-0.005153
Fuel	-0.111909	-0.049528	-0.132295	-0.298181	1.000000	0.021271	-0.292093	-0.078379
No of Owners	0.018655	0.000656	-0.483513	0.312665	0.021271	1.000000	-0.338083	-0.034446
Price	-0.005285	-0.092153	0.601707	-0.219021	-0.292093	-0.338083	1.000000	-0.002741
State	0.008898	0.030636	0.018801	-0.005153	-0.078379	-0.034446	-0.002741	1.000000

```
: sns.heatmap(df.corr())
  plt.show()
  #Heat map of correlation between columns
```
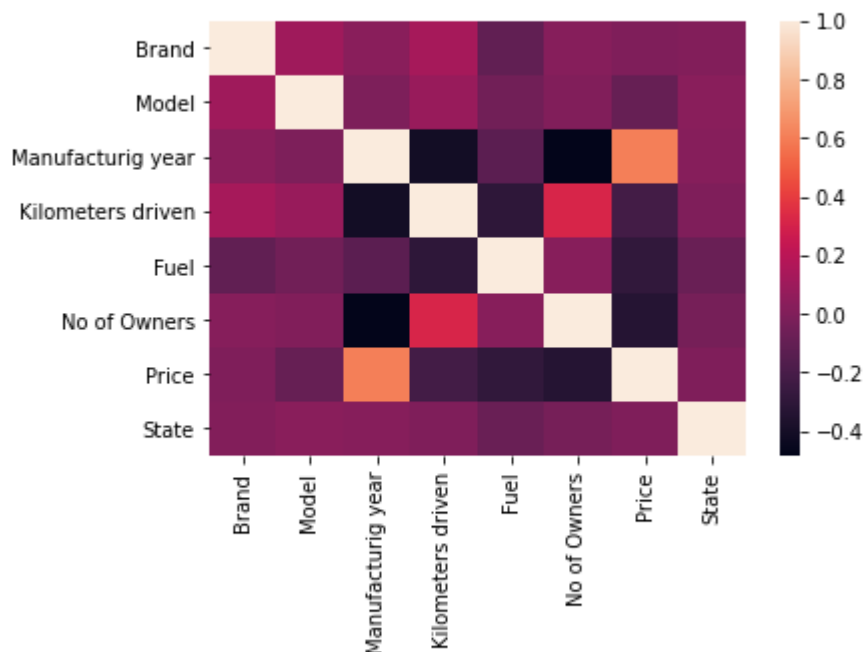

We can see that there is no multicorralation.

Splitting of data into Independent and Target variables

```
ind=df.drop("Price",axis=1)
tar=df["Price"]
#splitting individual and target variable in ind and tar
```

```
from sklearn.model_selection import train_test_split

# splitting data into training and testing
ind_train, ind_test, tar_train, tar_test = train_test_split(ind, tar, test_size=0.33, random_state=42)
```

- Data Inputs- Logic- Output Relationships

  In this project we can see that there are 9 columns namely "Brand", "Model", "Variant", "Manufacturing year", "Kilometers driven", "Fuel", "No of Owners", "Location" and "Price". Here "Price" is the target variable whereas the other 8 are independent variables.

  .

- State the set of assumptions (if any) related to the problem under consideration

  Here we have assumed that the "variant" column can be dropped as unique values are high. Also, we have dropped location by extracting only states to reduce unique values.

- Hardware and Software Requirements and Tools Used

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

#Importing Machine learning Models
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import AdaBoostRegressor

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  Describe the approaches you followed, both statistical and analytical, for solving of this problem.

- Testing of Identified Approaches (Algorithms)
  - DecisionTreeRegressor
  - RandomForestRegressor
  - LinearRegression
  - XGBRegressor
  - AdaBoostRegressor

- Run and Evaluate selected models

```python
for m in [rf,lr,xg,ad,dt]:
    m.fit(ind_train,tar_train)
    train_predict = m.predict(ind_train)
    test_predict = m.predict(ind_test)
    print('for',m)
    print("RMSE of train:", np.sqrt(mean_squared_error(tar_train, train_predict)))
    print("RMSE of test:", np.sqrt(mean_squared_error(tar_test, test_predict)))
    print("Train R^2: ", r2_score(tar_train, train_predict))
    print("Test R^2: ", r2_score(tar_test, test_predict))
    print("..........\n")
```

```
for RandomForestRegressor()
RMSE of train: 27.113481739416102
RMSE of test: 75.53616411645457
Train R^2:  0.9762729958535608
Test R^2:  0.8221423234277269
..........


for LinearRegression()
RMSE of train: 133.22525711955115
RMSE of test: 136.38377789940435
Train R^2:  0.42714400897223004
Test R^2:  0.4201866719070394
..........
```

```
for XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
RMSE of train: 22.955064326872662
RMSE of test: 67.44702623489738
Train R^2:  0.9829929357088604
Test R^2:   0.8581960352093563
..........

for AdaBoostRegressor(n_estimators=100, random_state=0)
RMSE of train: 128.30019991639577
RMSE of test: 134.18700835967067
Train R^2:  0.46871568881793235
Test R^2:   0.43871465573651824
..........

for DecisionTreeRegressor()
RMSE of train: 1.5547591000446013
RMSE of test: 103.11157639518838
Train R^2:  0.9999219813304426
Test R^2:   0.6685808384104553
..........
```

Crossvalidation:

```python
from sklearn.model_selection import cross_val_score
# function to get cross validation scores
def get_cv_scores(model):
    scores = cross_val_score(model,
                             ind_train,
                             tar_train,
                             cv=5,
                             scoring='r2')

    print('CV Mean: ', np.mean(scores))
    print('STD: ', np.std(scores))
    print('\n')
```

```python
for i in [rf,lr,xg,ad,dt]:
    print('for',i)
    print(get_cv_scores(i))
```

```
for RandomForestRegressor()
CV Mean:  0.8226862969021852
STD:  0.009232137204687978


None
for LinearRegression()
CV Mean:  0.42447176553805105
STD:  0.019002041821895507


None
for XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
CV Mean:  0.867971955780202
STD:  0.0071791614567836394


None
for AdaBoostRegressor(n_estimators=100, random_state=0)
CV Mean:  0.46670838955503874
STD:  0.022068279936120327

None
for DecisionTreeRegressor()
CV Mean:  0.6797734924948287
STD:  0.055965045725750145


None
```
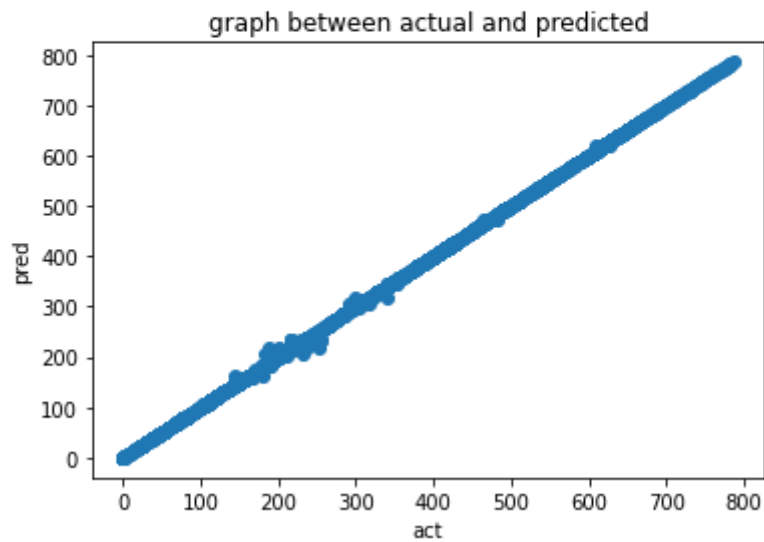
We can see that from the Accuracy score and Crossvalidation Score, XGBRegressor is performing the best. Therfore XGBRegressor is the best model

Graph between Actual and Predicted values

```
: plt.scatter(tar_train, train_predict)
  plt.title('graph between actual and predicted')
  plt.xlabel('act')
  plt.ylabel('pred')
  plt.show()
```


graph between actual and predicted

```
plt.scatter(tar_test, test_predict)
plt.title('graph between actual and predicted')
plt.xlabel('act')
plt.ylabel('pred')
plt.show()
```


graph between actual and predicted
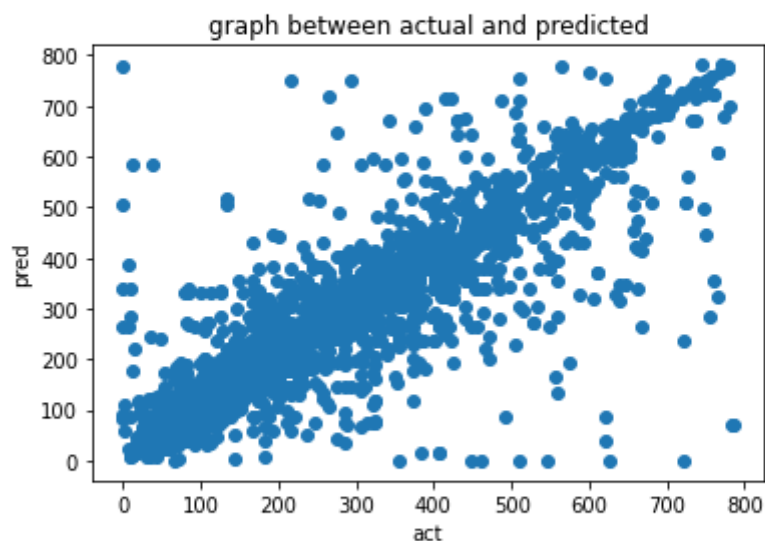
We can see that the above two graphs that the model is predicting well.

Hyperparameter Tuning

```python
# To find the best parameters by using GridSearchCV to improve the model
from sklearn.model_selection import GridSearchCV

parameters = {'min_child_weight':[4,5], 'gamma':[i/10.0 for i in range(3,6)],
              'subsample':[i/10.0 for i in range(6,11)],
            'colsample_bytree':[i/10.0 for i in range(6,11)], 'max_depth': [2,3,4]}

gs=GridSearchCV(xg,parameters)
gs.fit(ind_train,tar_train)
print(gs.best_params_)
#best parameters
```

{'colsample_bytree': 1.0, 'gamma': 0.3, 'max_depth': 4, 'min_child_weight': 5, 'subsample': 1.0}

```python
gs.best_score_
```

0.8695142800886895

We can see that the score is good. Therefore the model is predicting well.

- ## Key Metrics for success in solving problem under consideration
    a. Accuracy score
    b. Crossvalidation score
    c. Hyperparameter tuning

- ## Interpretation of the Results

We can see from the above results that the best accuracy and cross-validation score is for XGBRegressor Model. Therefore, we can say that XGBRegressor is the best model among them.

CONCLUSION

- ## Key Findings and Conclusions of the Study

We have found that Logistic XGBRegressor model is the model that works best with this data among the other 4.

- ## Learning Outcomes of the Study in respect of Data Science

Cleaning of the data is very important, also making them in the format they represent as it would affects the model training and thereby predict wrong results. We can eliminate features that have more no of unique values by keeping the importance of the feature in mind.

- ## Limitations of this work and Scope for Future Work

The only limitation I have found is that we need to keep updating or making new models as condition change and few more features keep adding up. Therefore, we need to keep updating the model by using the new dataset.