Predicting Used Car Prices using Machine Learning.

BY

BHARGAVA RATHOD

SAIPRATHYUSHA VEGURU

SHESHIDHAR REDDY SHAGA

LIKITH CHOWDARY

MEGHANA ERAVELLI

Abstract

- In this project, we are going to train the Regression Model that will help us predict used car prices. We will apply some machine learning techniques for the price of cars with the available independent variables. That should help the management or individual to understand how exactly the prices vary with the independent variables.

Dataset

- The dataset has in total of 301 rows and 9 columns.
- Columns such as: Car name, Year, Selling Price, Present price, Kms driven, Fuel type, Seller type, Transmission, Owners.
- Dataset is divided into train (80%) and test (20%).

I have used Jupyter Notebooks to train the model.

Data Preprocessing.

- Out of 9 columns there are 4 columns that are string data type.
- Since they are categorical, we have used label encoder to convert the values of Fuel type, seller type, transmission into numerical values.
- Whereas there are too many categories for car name and kms_driven We have used On hot encoding for car name and standard scaler for kms_driven.

```
In [31]: columns
Out[31]: ['Present_Price',
          'Fuel_Type',
          'Seller_Type',
          'Transmission',
          'Owner',
          'num_years',
          '800',
          'activa_3g',
          'activa_4g',
          'alto_800',
          'alto_k10',
          'amaze',
          'bajaj__ct_100',
          'bajaj_avenger_150',
          'bajaj_avenger_150_street',
          'bajaj_avenger_220',
          'bajaj_avenger_220_dtsi',
          'bajaj_avenger_street_220',
          'bajaj_discover_100',
          'bajaj_discover_125',
          'bajaj_dominar_400',
          'bajaj_pulsar_135_ls',
          'bajaj_pulsar_150',
          'bajaj_pulsar_220_f',
          'bajaj_pulsar_ns_200',
          'bajaj_pulsar_rs200',
          'baleno',
          'brio',
          'camry',
          'ciaz',
          'city',
          'corolla',
          'corolla_altis',
          'creta',
          'dzire',
          'elantra',
          'eon',
          'ertiga',
          'etios_cross',
          'etios_g',
          'etios_liva',
          'fortuner',
          'grand_i10',
```

Screenshot

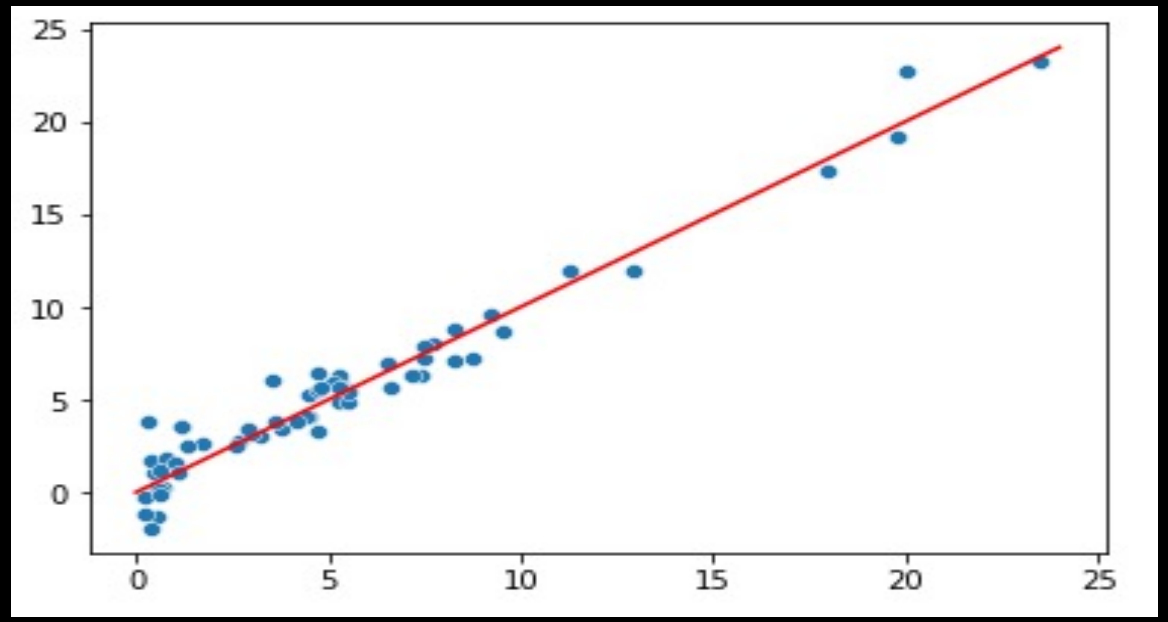
```
In [14]: data['Transmission'].replace({'Manual':0,'Automatic':1},inplace=True)
         data.head()
```

Out[14]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	0	0	0	0
1	sx4	2013	4.75	9.54	43000	1	0	0	0
2	ciaz	2017	7.25	9.85	6900	0	0	0	0
3	wagon r	2011	2.85	4.15	5200	0	0	0	0
4	swift	2014	4.60	6.87	42450	1	0	0	0

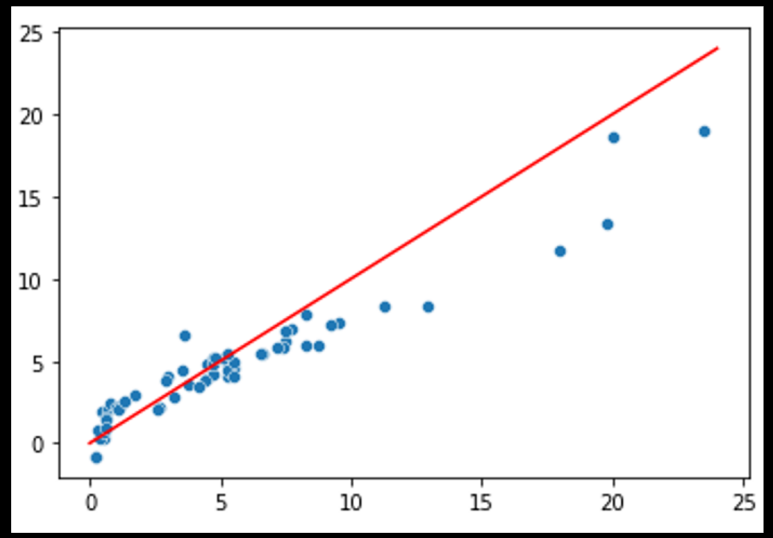
```
In [28]: X_train_Kms_Driven[:10],X_test_Kms_Driven[:10]
Out[28]: (array([[ 1.05706025],
                 [-0.3529268 ],
                 [-0.77406088],
                 [-0.13152761],
                 [-0.23735214],
                 [ 0.51190933],
                 [ 0.55766358],
                 [-0.88583061],
                 [ 0.1058285 ],
                 [-0.32222579]]),
          array([[ 0.54294327],
                 [-0.77406088],
                 [-0.32988321],
                 [-0.10848402],
                 [-0.56003373],
                 [-0.78357319],
                 [-0.08441785],
                 [-0.89296485],
                 [-0.15576024],
                 [-0.05825898]]))
```

Model 1 (linear regression)




```
In [34]: lr = LinearRegression()
         lr.fit(X_tr,y_train)

Out[34]: LinearRegression()

In [35]: predicted = lr.predict(X_te)

In [36]: mean_squared_error(y_test,predicted)

Out[36]: 1.2348648058028 13
```

Model 2 (RandomizedSearchCV using lasso regression)

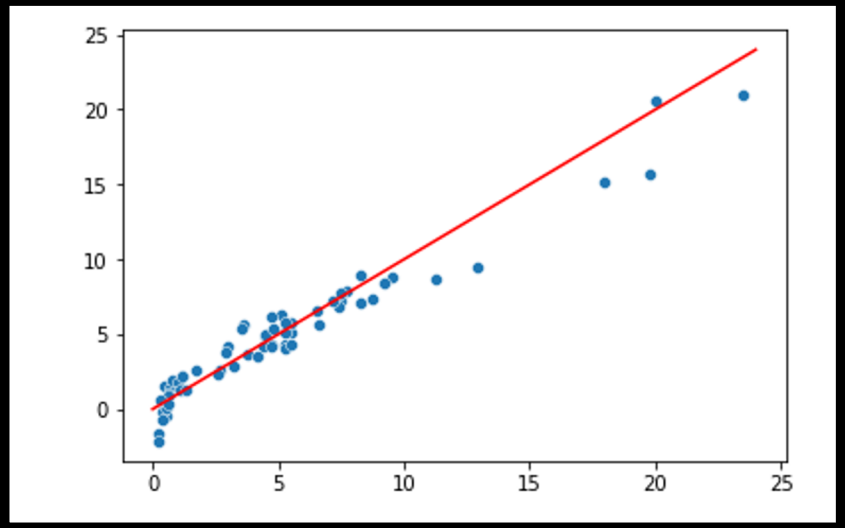



```
alphas = {'alpha': [5, 0.5, 0.05, 0.005, 0.0005, 1, 0.1, 0.01, 0.001, 0.0001, 0 ]}
```

```
ls = Lasso()
RS_CV = RandomizedSearchCV(estimator = ls, param_distributions = alphas,scoring='neg_mean_squared_error', n_iter = 10,
RS_CV.fit(X_tr, y_train)
```

```
In [41]: lasso_model = RS_CV.best_estimator_
         predicted = lasso_model.predict(X_te)

In [42]: mean_squared_error(y_test,predicted)
Out[42]: 3.3332706945534243
```

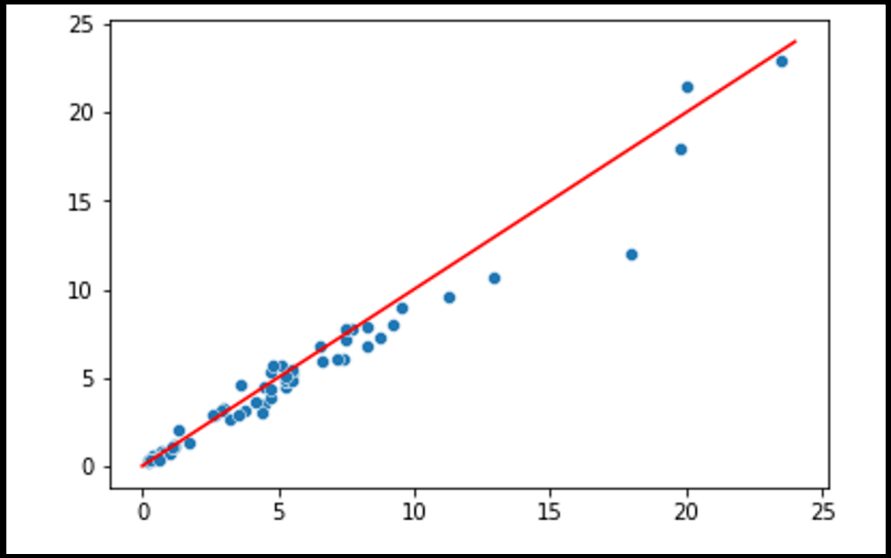
Model 3(RandomizedSearchCV using ridge regression)




```
rd = Ridge()

RS_CV = RandomizedSearchCV(estimator = rd, param_distributions = alphas,scoring='neg_mean_squared_error', n_iter = 10,
RS_CV.fit(X_tr, y_train)
```

```
[46]: ridge_model = RS_CV.best_estimator_
      predicted = ridge_model.predict(X_te)

[47]: mean_squared_error(y_test,predicted)

[47]: 1.5284677389896668
```

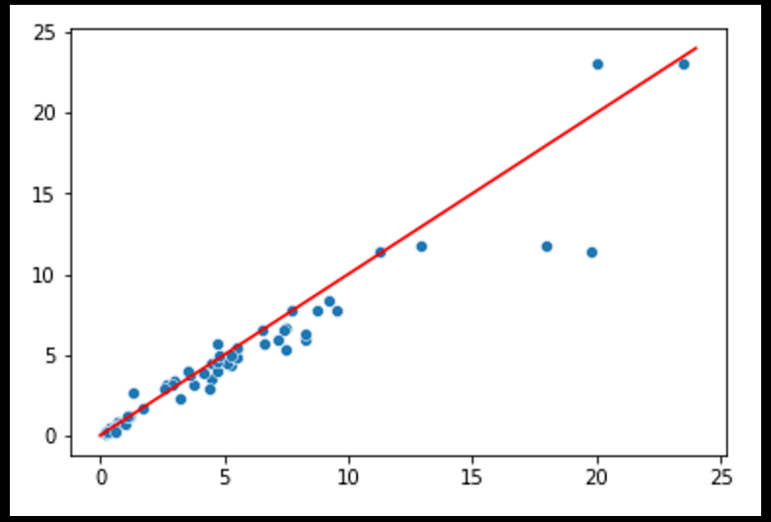
Model 4
KNeighborsRegressor




```
knnR = KNeighborsRegressor()

params = { 'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}

RS_CV = RandomizedSearchCV(estimator = knnR, param_distributions = params,scoring='neg_mean_squared_error', n_iter = 10
RS_CV.fit(X_tr, y_train)
```

```
51]: KNN_reg_model = RS_CV.best_estimator_
     predicted = KNN_reg_model.predict(X_te)

52]: mean_squared_error(y_test,predicted)

52]: 1.1520370852916368
```

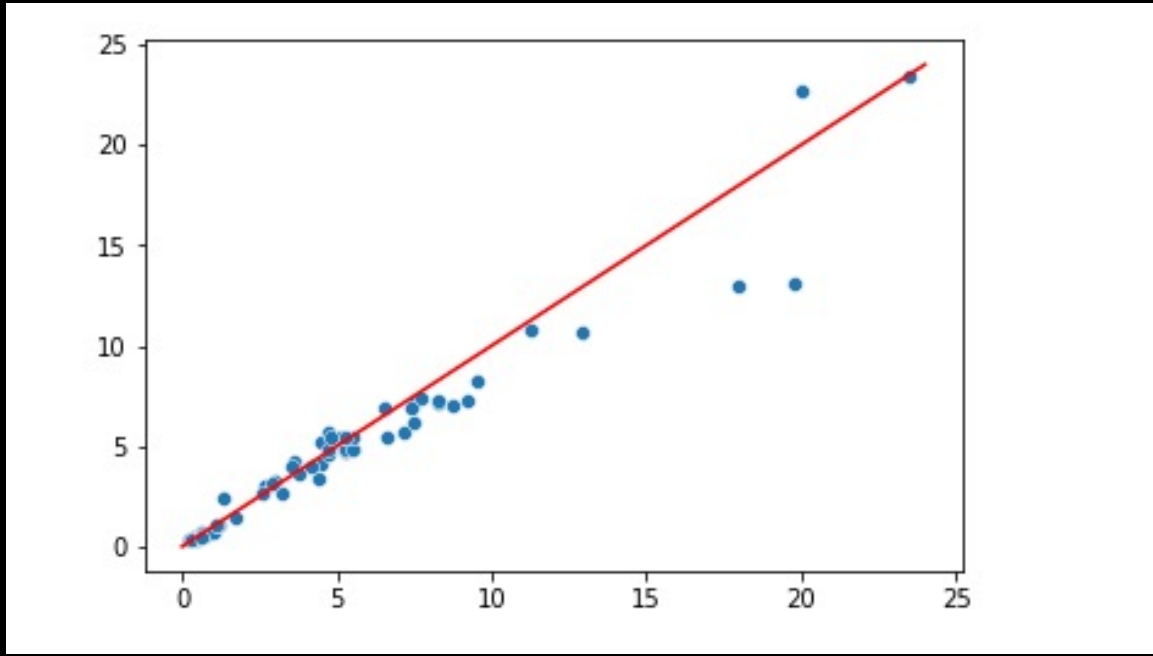
Model 5
DecisionTreeRegressor




```
]: params={"max_depth" : [1,3,5,7,9,11,12],
          "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
          "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90] }

RS_CV = RandomizedSearchCV(estimator = dtr, param_distributions = params,scoring='neg_mean_squared_error', n_iter = 10,
RS_CV.fit(X_tr, y_train)
```

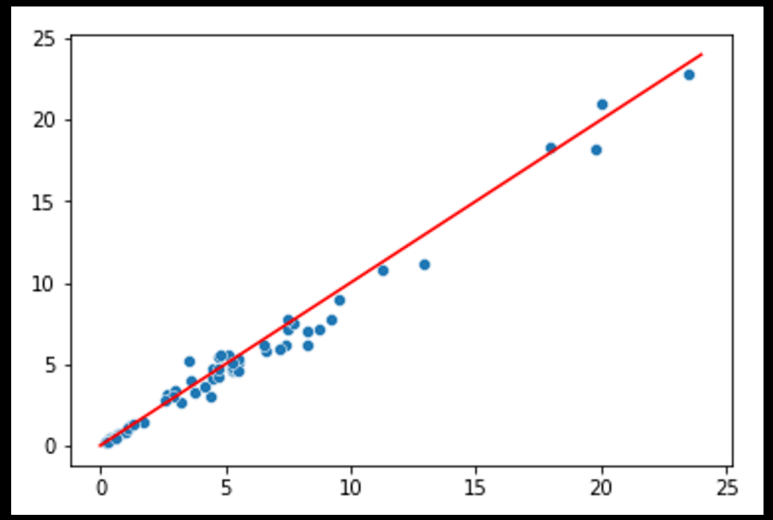
```
[57]: decision_tree_r_model = RS_CV.best_estimator_
      predicted = decision_tree_r_model.predict(X_te)

[58]: mean_squared_error(y_test,predicted)

[58]: 2.495222576502733
```

Model 6 RandomForestRegressor




```
]: RandomForest_r_model = RS_CV.best_estimator_
   predicted = RandomForest_r_model.predict(X_te)

]: mean_squared_error(y_test,predicted)

]: 1.7221434508757818
```

Model 7 ExtraTreesRegressor




```
etr = ExtraTreesRegressor()

params={"max_depth" : [1,3,5,7,9,11,12],
        "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
        "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]}

RS_CV = RandomizedSearchCV(estimator = etr, param_distributions = params,scoring='neg_mean_squared_error', n_iter = 10
RS_CV.fit(X_tr, y_train)
```

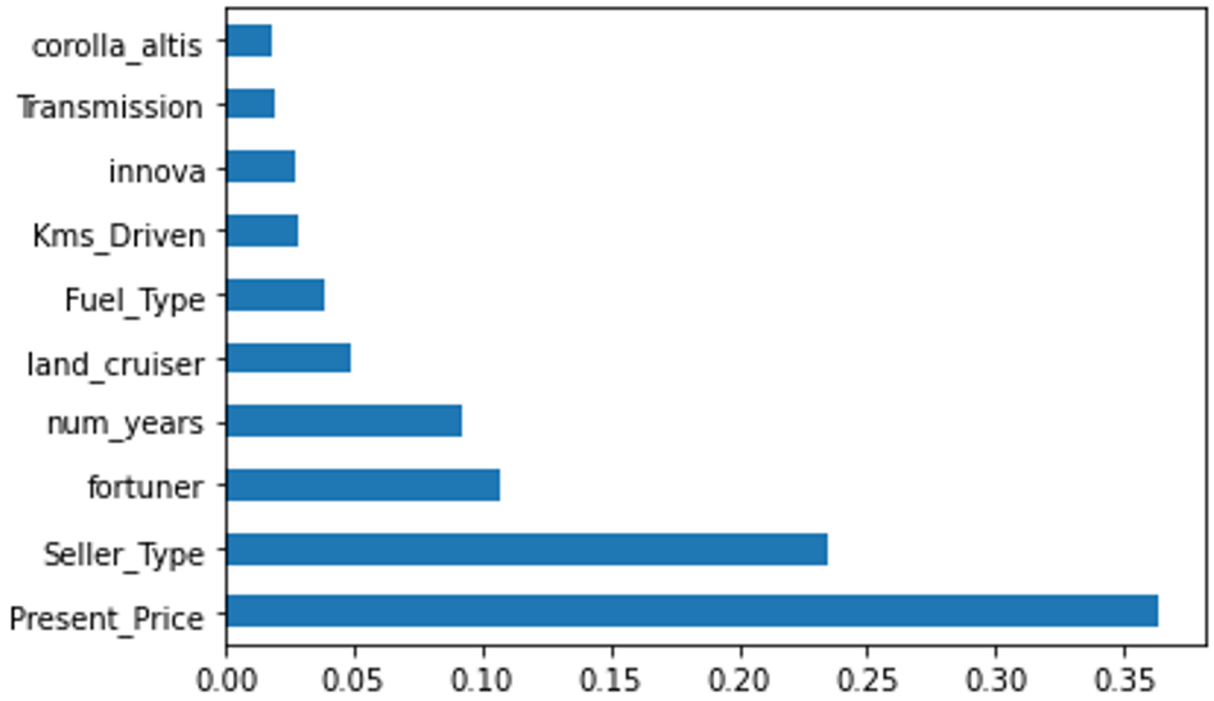
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END .max_depth=3, max_leaf_nodes=50, min_samples_leaf=9; total time=   0.1s
[CV] END .max_depth=3, max_leaf_nodes=50, min_samples_leaf=9; total time=   0.1s
[CV] END .max_depth=3, max_leaf_nodes=50, min_samples_leaf=9; total time=   0.0s
[CV] END .max_depth=3, max_leaf_nodes=50, min_samples_leaf=9; total time=   0.1s
```

```
ExtraTrees_model = RS_CV.best_estimator_
predicted = ExtraTrees_model.predict(X_te)
```

```
mean_squared_error(y_test,predicted)
```

```
0.5238840116260965
```

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(ExtraTrees_model.feature_importances_, index=columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



Future enhancements

- Search any large dataset that consists of Present price column.
- Use other regression models if we can minimize the mean square error that the one we have achieved.