UsedCar-Price Prediction Project

Submitted by:

SARMISHTHA HALDAR

ACKNOWLEDGEMENT:

I would like to thank all my teachers, supervisors for the learning. Few journals referred in the case are as follows:

i) Gareth, J., Daniela, W., Trevor, H., & Tibshirani, R. (2013). *An Introduction to Statistical*

ii) Raschka, S., & Mirjalili, V. (2017). *Python machine learning*. Packt Publishing Ltd.

Business Problem Framing

This is a classic Business problem which helps to evaluate the price of the used car using the modelling below. The problem has occurred due to recent changes in the car market due to COVID-19 impact.

Conceptual Background

With 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 and hence making them cheaper. With the change in market due to covid-19 impact the previous price evaluation models are not serving the purpose and hence we need to provide a car evaluation model which will help them to decide the car prices.

INTRODUCTION

Determining whether the listed price of a used car 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, in order to make informed purchases. We implement and evaluate various learning methods on a dataset consisting of the sale prices of different makes and models across cities in India.

Problem Statement

To Build a model which can be used to predict prices of used cars

Analytical Problem Framing

We have used methods like r2score and RMSE for model evaluations

R^2 is a statistic that will give some information about the goodness of fit of a model. In regression, the R^2 **coefficient of determination** is a statistical measure of how well the regression predictions approximate the real data points. An R^2 of 1 indicates that the regression predictions perfectly fit the data

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

- Data Sources and their formats

  We received the data in the form of .csv file and data was loaded using Pandas

n [2]:
```
Car=pd.read_csv('Cardetails.csv')
```

n [3]:
```
Car.head()
```

ut[3]:

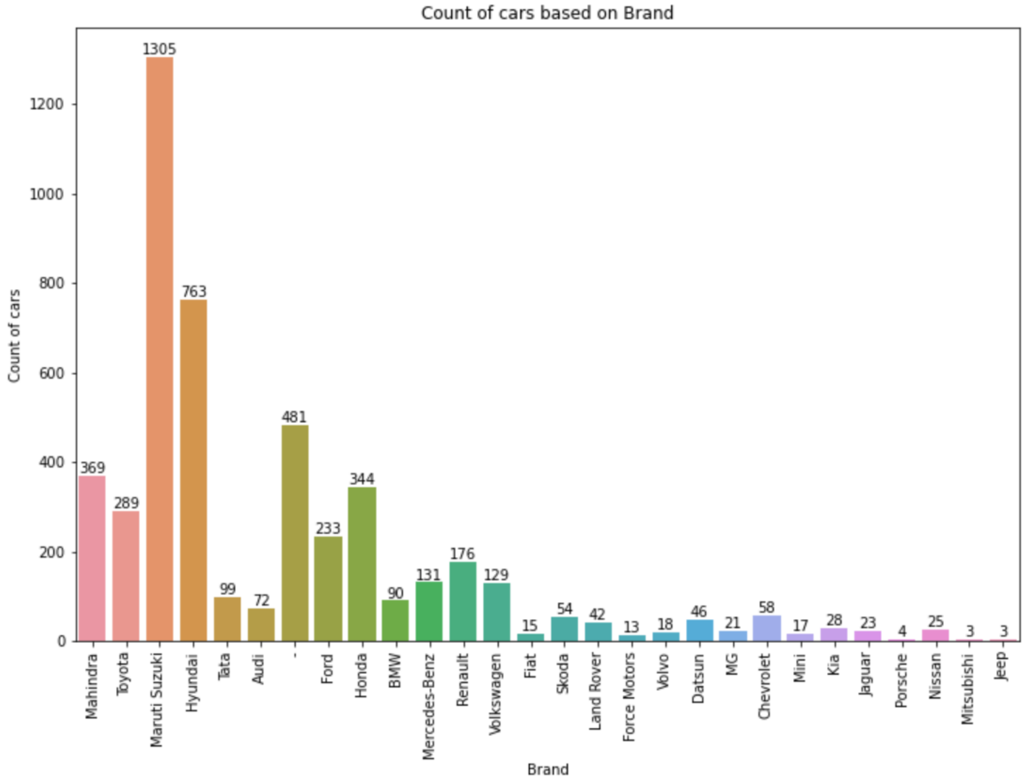
	Unnamed: 0	Brand	Model	Variant	Manufacturing_Year	Driven_kilometres	Fuel	Number_of_owners	Location	Price
0	0	Maruti Suzuki	Celerio	ZXI AMT	2017	11,439 km	Petrol	1st	Sainikpuri, Hyderabad, Telangana	₹ 4,90,000
1	1	Mahindra	Xylo	2009-2011 E8	2011	81,000 km	Diesel	1st	Rocktown Colony, Hyderabad, Telangana	₹ 3,96,000
2	2	Tata	Nexon	1.2 Revotron XM	2018	55,700 km	Petrol	1st	Himayat Nagar, Hyderabad, Telangana	₹ 7,75,000
3	3	Honda	CR-V	2007-2012 AT With Sun Roof	2010	71,174 km	Petrol	1st	Madhapur, Hyderabad, Telangana	₹ 7,50,000
4	4	Maruti Suzuki	Swift Dzire	VDI	2019	65,035 km	Diesel	1st	Ameerpet, Hyderabad, Telangana	₹ 7,90,000

- Exploratory Data Analysis

i)Brand

In [17]:
```
Car['Brand'].value_counts()
```

Out[17]:
```
Maruti Suzuki    2570
Hyundai          1090
Mahindra          530
Honda             478
Toyota            428
Ford              329
Renault           232
Mercedes-Benz     179
Volkswagen        175
Tata              162
BMW               129
Audi               93
Chevrolet          86
Skoda              74
Land Rover         72
Datsun             65
Kia                45
Jaguar             36
Volvo              31
Nissan             28
MG                 28
Fiat               20
Force Motors       18
Mini               18
Mitsubishi          6
Jeep                5
Porsche             4
Name: Brand, dtype: int64
```



Maximum cars in the dataset are by the manufacturer Maruti and looks like its quite popular .

- Location

```
In [19]: Car['Location'].value_counts()
Out[19]: -                                                               706
         Pitampura, Delhi, Delhi                                         229
         Noida Extension, Noida, Uttar Pradesh                           171
         Madhapur, Hyderabad, Telangana                                  146
         Hazratganj, Lucknow, Uttar Pradesh                              137
                                                                        ...
         Infocity, Gandhinagar, Gujarat                                    3
         Subhash Park, Vadodara, Gujarat                                   2
         Bhagwan Nagar Tekra, Ahmedabad, Gujarat                           2
         Billekahalli, Bengaluru, Karnataka                                2
         Banaswadi Rammurthi Nagar Green Park Layout, Bengaluru, Karnataka 2
         Name: Location, Length: 260, dtype: int64
```

There are few missing values and we can look at it while data preprocessing

- **Driven_kilometres**

```
In [34]: X_train["Driven_kilometres"]

Out[34]: 3203     71,000 km
         1350    110,000 km
         6812     67,000 km
         446     175,835 km
         1743     61,000 km
                    ...
         3772     15,000 km
         5191     90,000 km
         5226     39,000 km
         5390     39,000 km
         860      44,000 km
         Name: Driven_kilometres, Length: 4851, dtype: object
```

This clearly shows that data range is really high and high values might affect prediction thus it is important that scaling can be applied .

- **Year**

This simply displays the year which we have applied function to calculate the age of the car.

- Data Preprocessing

i)Removing the Unwanted columns

Remove the unwanted columns

```
In [4]: Car.drop('Unnamed: 0',inplace=True,axis=1)

In [5]: Car.head()
```

Out[5]:

	Brand	Model	Variant	Manufacturing_Year	Driven_kilometres	Fuel	Number_of_owners	Location	Price
0	Maruti Suzuki	Celerio	ZXI AMT	2017	11,439 km	Petrol	1st	Sainikpuri, Hyderabad, Telangana	₹ 4,90,000
1	Mahindra	Xylo	2009-2011 E8	2011	81,000 km	Diesel	1st	Rocktown Colony, Hyderabad, Telangana	₹ 3,96,000
2	Tata	Nexon	1.2 Revotron XM	2018	55,700 km	Petrol	1st	Himayat Nagar, Hyderabad, Telangana	₹ 7,75,000
3	Honda	CR-V	2007-2012 AT With Sun Roof	2010	71,174 km	Petrol	1st	Madhapur, Hyderabad, Telangana	₹ 7,50,000
4	Maruti Suzuki	Swift Dzire	VDI	2019	65,035 km	Diesel	1st	Ameerpet, Hyderabad, Telangana	₹ 7,90,000

```
In [6]: Car['Price'] = Car['Price'].replace('\₹', '', regex=True)
```

In [20]:
```
Car['Location']=Car['Location'].replace('-','Delhi')
```

In [21]:
```
Car.Location.mode()
```

Out[21]:
```
0    Delhi
dtype: object
```

Location should not be a determinant for the price of a car and I'll safely remove it.

ii) Checking for null/missing values and imputing them with mean,mode, median as required

i)Dealing with missing values

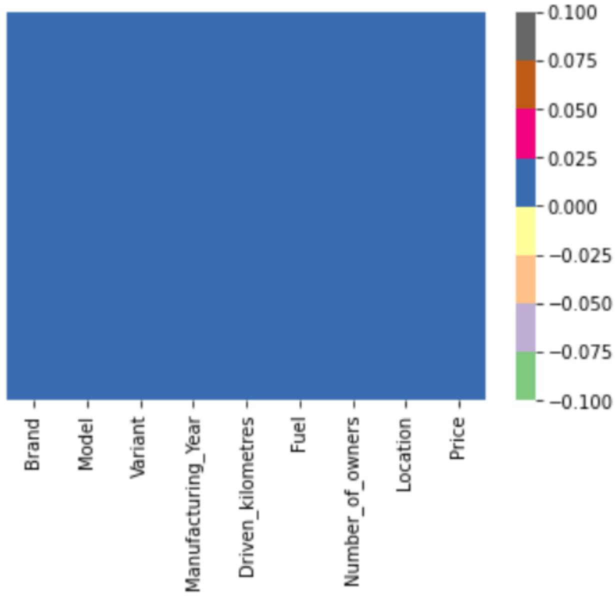
In [35]:
```
Car.isnull().sum()
```

Out[35]:
```
Brand                  0
Model                  0
Variant                0
Manufacturing_Year     0
Driven_kilometres      0
Fuel                   0
Number_of_owners       0
Location               0
Price                  0
dtype: int64
```

In [36]:
```
#There are no missing values in the dataframe
```

In [37]:
```
#heatmap to verify nulll values using graph
sns.heatmap(Car.isnull(),yticklabels=False,cbar=True,cmap='Accent')
```

Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x265bf665280>
```



iii) Converting the columns to required data types and meaningful data

Year

```
In [24]: curr_time = datetime.datetime.now()
```

```
In [31]: X_train['Manufacturing_Year']=X_train['Manufacturing_Year'].astype(int)
         X_test['Manufacturing_Year'] = X_test['Manufacturing_Year'].astype(int)
```

```
In [32]: X_train['Manufacturing_Year'] = X_train['Manufacturing_Year'].apply(lambda x : curr_time.year - x)
         X_test['Manufacturing_Year'] = X_test['Manufacturing_Year'].apply(lambda x : curr_time.year - x)
```

```
In [33]: X_train['Manufacturing_Year']

Out[33]: 3203     9
         1350     7
         6812    10
         446      7
         1743     9
                 ..
         3772     3
         5191    11
         5226    20
         5390    10
         860      5
         Name: Manufacturing_Year, Length: 4851, dtype: int64
```

Model,Variant,Fuel,Number_of_owners .All these columns are categorical columns which should be converted to dummy variables before being used

```
In [38]: #define numeric variable and categorical variable to work separatly on them
         num_col=['Manufacturing_Year']
         cat_cols=['Brand','Model','Variant','Fuel','Number_of_owners','Location','Driven_kilometres']
```

Now that we have worked with the training data, let's create dummy columns for categorical columns before we begin training.

```
In [39]: X_train = pd.get_dummies(X_train,
                                  columns = ["Brand", "Model", "Variant","Fuel", "Number_of_owners","Driven_kilometres"],
                                  drop_first = True)
```

```
In [40]: X_test = pd.get_dummies(X_test,
                                  columns = ["Brand", "Model", "Variant","Fuel", "Number_of_owners","Driven_kilometres"],
                                  drop_first = True)
```

```
In [41]: #It might be possible
         #that the dummy column creation would be different in test and train data, thus, I'd fill in all missing columns with zeros.
```

```
In [42]: missing_cols = set(X_train.columns) - set(X_test.columns)
         for col in missing_cols:
             X_test[col] = 0
         X_test = X_test[X_train.columns]
```

Iv) Scaling the data

```
In [43]: standardScaler = StandardScaler()

In [44]: standardScaler.fit(X_train)

Out[44]: StandardScaler()

In [45]: X_train = standardScaler.transform(X_train)
         X_test = standardScaler.transform(X_test)
```

- Hardware and Software Requirements , Tools Used

  No Specific requirements except Jupyter Notebook.

Model/s Development and Evaluation

- Identification of possible problem

  This is a Regression Problem and we have used Decision tree, Random Forest,KNN ,LASSO,XGboost,Gradientboosting regressor to build the model.

- Run and Evaluate selected models:We used Linear regression,Decisiontree regressor ,Randomforest Regressor and found that Decisiontree and random forest are performing well when we found the r2 score. Then we further did a gridsearch tuning with host of other models and then created an average of the best performing models.

- ## KNNRidgeregression:
  KNN has been used in **statistical estimation and pattern recognition** already in the beginning of 1970's as a non-parametric technique. A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighb ors

```
In [56]: KRR = KernelRidge()

         KRR_grid = {"alpha" : [25,10,4,2,1.0,0.8,0.5,0.3,0.2,0.1,0.05,0.02,0.01],
                     "kernel" :   ["polynomial"],
                     "degree" : [1,2,3,4,5],
                     "coef0" :[1,1.5,2,2.5,3,3.5,4,4.5,5]
                     }

         KRRModel = GridSearchCV(estimator = KRR, param_grid = KRR_grid, cv=kf, scoring="neg_mean_squared_error", n_jobs= 4, verbose = 1)
         KRRModel.fit(X_train,y_train)
         KRR_best = KRRModel.best_estimator_
         KRRModel.best_params_

         Fitting 5 folds for each of 585 candidates, totalling 2925 fits
Out[56]: {'alpha': 0.01, 'coef0': 3, 'degree': 2, 'kernel': 'polynomial'}
```

Gradient boosting regressor: GradientBoostingRegressor GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

```
In [66]: GBR = GradientBoostingRegressor()

         GBR_grid = {"n_estimators" : [2000,3000],
                     "learning_rate" :   [0.01,0.1],
                     "max_depth" : [3,5],
                     "max_features" :['sqrt'],
                     "min_samples_leaf" :[10,15],
                     "min_samples_split" :[2,5],
                     "loss" :['huber']
                     }

         GBRModel = GridSearchCV(estimator = GBR, param_grid = GBR_grid, cv=kf, scoring="neg_mean_squared_error", n_jobs= 4, verbose = 1)
         GBRModel.fit(train,y_train)
         GBR_best = GBRModel.best_estimator_
         GBRModel.best_params_

         Fitting 5 folds for each of 32 candidates, totalling 160 fits
Out[66]: {'learning_rate': 0.01,
          'loss': 'huber',
          'max_depth': 3,
          'max_features': 'sqrt',
          'min_samples_leaf': 15,
          'min_samples_split': 5,
          'n_estimators': 3000}
```

XGBRegressor: It is an efficient implementation of gradient boosting that can be used for regression predictive modeling

```
In [67]: XGB = XGBRegressor()

         XGB_grid =    {'nthread':[4],
                        'objective':['reg:linear'],
                        'learning_rate': [.03, 0.05, .07],
                        'max_depth': [5, 6, 7],
                        'min_child_weight': [4],
                        'silent': [1],
                       'subsample': [0.7],
                        'colsample_bytree': [0.7],
                       'n_estimators': [500]}

         XGBModel = GridSearchCV(estimator = XGB, param_grid = XGB_grid, cv=kf, scoring="neg_mean_squared_error", n_jobs= 4, verbose = 1)
         XGBModel.fit(train,y_train)
         XGB_best = XGBModel.best_estimator_
         XGBModel.best_params_
```

```
Out[67]: {'colsample_bytree': 0.7,
          'learning_rate': 0.07,
          'max_depth': 5,
          'min_child_weight': 4,
          'n_estimators': 500,
          'nthread': 4,
          'objective': 'reg:linear',
          'silent': 1,
          'subsample': 0.7}
```

Decision Tree Regressor: Decision Tree - Regression. Decision tree builds regression or classification models in the form of a tree structure. **It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed**. The final result is a tree with decision nodes and leaf nodes.

```
In [59]: DTR = DecisionTreeRegressor(random_state=0)

         DTR_grid =    {"criterion":['mse'],
                         "splitter":['best'],
                         "max_depth" : [2,3,5,10],
                      "max_features" :['sqrt'],
                      "min_samples_leaf" :[5,10,15],
                      "min_samples_split" :[1,2,5]
                        }

         DTRModel = GridSearchCV(estimator = DTR, param_grid = DTR_grid, cv=kf, scoring="neg_mean_squared_error", n_jobs= 4, verbose = 1)
         DTRModel.fit(X_train,y_train)
         DTR_best = DTRModel.best_estimator_
         DTRModel.best_params_

         Fitting 5 folds for each of 36 candidates, totalling 180 fits

Out[59]: {'criterion': 'mse',
          'max_depth': 10,
          'max_features': 'sqrt',
          'min_samples_leaf': 5,
          'min_samples_split': 2,
          'splitter': 'best'}
```

LASSO: Lasso regression is **a regularization technique**. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters).

```
In [60]: LARS = Lasso()

         LARS_grid = {"alpha" : [1,0.8,0.3,0.2,0.1,0.05,0.005,0.02,0.01],
                      "max_iter" :[500,700,1000]
                          }

         LARSModel = GridSearchCV(estimator = LARS, param_grid = LARS_grid, cv=kf, scoring="neg_mean_squared_error", n_jobs= 4, verbose = 1)
         LARSModel.fit(X_train,y_train)
         LARS_best = LARSModel.best_estimator_
         LARSModel.best_params_

         Fitting 5 folds for each of 27 candidates, totalling 135 fits

Out[60]: {'alpha': 1, 'max_iter': 1000}
```

In [61]:
```
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(X_train)
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

In [62]:
```
#Build the base models based on GridSearch tuning

KRR = KernelRidge(alpha=0.8, coef0=5, degree=2, gamma=None, kernel='polynomial', kernel_params=None)

lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, max_iter = 500, random_state=1))

GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.01, max_depth=3, max_features='sqrt', min_samples_leaf=10, min_sa
mples_split=5, loss='huber')

model_xgb = xgb.XGBRegressor(colsample_bytree=0.7, learning_rate=0.03, max_depth=6, min_child_weight=4, n_estimators=500, subsample=0.
7, silent=1, random_state =7)

model_DTR = DecisionTreeRegressor(criterion = 'mse', max_depth = 10, max_features = 'sqrt', min_samples_leaf = 5, min_samples_split =
2, splitter = 'best' )

LRModel = LinearRegression()
```

[64]:
```
score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = rmsle_cv(lasso)
print("Lasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = rmsle_cv(model_DTR)
print("DT Regression score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))

score = rmsle_cv(LRModel)
print("Linear Regression score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))
```

```
Kernel Ridge score: 36216.4901 (12173.1580)

Lasso score: 29603.2268 (9415.6308)

Gradient Boosting score: 359391.7890 (30830.6505)

Xgboost score: 131510.8550 (13899.1997)

DT Regression score: 644898.8933 (45017.9086)

Linear Regression score: 805740048700595200.0000 (330606285470877888.0000)
```

Averaging the best models to create a model

```
In [65]: class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
             def __init__(self, models):
                 self.models = models

             # we define clones of the original models to fit the data in
             def fit(self, X, y):
                 self.models_ = [clone(x) for x in self.models]

                 # Train cloned base models
                 for model in self.models_:
                     model.fit(X, y)

                 return self

             #Now we do the predictions for cloned models and average them
             def predict(self, X):
                 predictions = np.column_stack([
                     model.predict(X) for model in self.models_
                 ])
                 return np.mean(predictions, axis=1)
```

```
In [66]: #Averaging the best models to optimize the prediction.

         #averaged_models = AveragingModels(models = (KRR, lasso, ENet, GBoost, model_xgb, model_lgb, model_DTR, LRModel))
         averaged_models = AveragingModels(models = (KRR,GBoost,model_DTR, lasso, model_xgb))

         score = rmsle_cv(averaged_models)
         print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Averaged base models score: 210733.5443 (16892.4285)
```

```
]: averaged_models.fit(X_train, y_train)
   averaged_models_pred = np.expm1(averaged_models.predict(X_test))
```

- Key Metrics for success in solving problem under consideration


```
In [65]: class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
             def __init__(self, models):
                 self.models = models

             # we define clones of the original models to fit the data in
             def fit(self, X, y):
                 self.models_ = [clone(x) for x in self.models]

                 # Train cloned base models
                 for model in self.models_:
                     model.fit(X, y)

                 return self

             #Now we do the predictions for cloned models and average them
             def predict(self, X):
                 predictions = np.column_stack([
                     model.predict(X) for model in self.models_
                 ])
                 return np.mean(predictions, axis=1)
```

```
In [66]: #Averaging the best models to optimize the prediction.

         #averaged_models = AveragingModels(models = (KRR, lasso, ENet, GBoost, model_xgb, model_lgb, model_DTR, LRModel))
         averaged_models = AveragingModels(models = (KRR,GBoost,model_DTR, lasso, model_xgb))

         score = rmsle_cv(averaged_models)
         print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Averaged base models score: 210733.5443 (16892.4285)
```

```
]: averaged_models.fit(X_train, y_train)
   averaged_models_pred = np.expm1(averaged_models.predict(X_test))
```


CONCLUSION

- ## Conclusions

  According to the performance metrics, Lasso, Decisiontree regressor,KNN,Gradient boosting have good scores and hence we build a model by averaging the best ones as above.

- ## Limitations & Scope for Future

This study used different models in order to predict used car prices. However, there was a relatively small dataset for making a strong inference because number of observations was only 7000. Gathering more data can yield more robust predictions. Secondly, there could be more features that can be good predictors. For example, here are some variables that might improve the model: number of doors, gas/mile (per gallon), color, mechanical and cosmeticreconditioning time, used-to-new ratio, appraisal-to-trade ratio.

Another point that that has room to improvement is that data cleaning process can be dome more rigorously with the help of more technical information