Old Cars Resale Price Prediction

Submitted By
Swati Kumari

July 11, 2022

Agenda:

- Data Scraping
- Data Preparation
- Explanatory Data Analysis
- Data Cleaning & Imputation
- Remove Outliers
- Feature engineering
- Model selection
- Model Prediction
- Conclusion

Data Scraping

- Old cars prices are scraped from online websites like Cars24, cardekho.com and Olx.
- Data is scraped for major cities like Delhi, Noida, Gurgaon, Bangalore, Mumbai, Pune, Ahmedabad, Hyderabad, Kolkata etc.
- Different features like Brand and Model Name, car variant, year of manufacturing, Fuel type, Owner type, Transmission, Kilometer driven, city(location) and selling price were scraped for each car.
- Around 5058 data were scraped for 9 different features.
- Some data cleaning was done.
- The data frame was made and changed into CSV file.
- The final dataset has 5058 rows and 9 features which was further loaded for EDA and Machine learning.

Data Preprocessing

- Checking the shape of Datasets
- Checking the columns
- Checking the Data types Of independent features
- Checking the null values
- Checking and dropping the unwanted columns
- Checking Categorical columns and numerical columns

Checking the datatype

```
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5058 entries, 0 to 5057
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Brand Name         5058 non-null   object
 1   Year               5058 non-null   int64
 2   Car_variant        5058 non-null   object
 3   Selling Price      5058 non-null   int64
 4   Kilometers_Driven  5058 non-null   int64
 5   Fuel_Type          5058 non-null   object
 6   Transmission       5002 non-null   object
 7   Owner_Type         4898 non-null   object
 8   Location           5058 non-null   object
dtypes: int64(3), object(6)
memory usage: 355.8+ KB
```

- The dataset has 3 numerical columns out of which Selling price is also one which is target column. Rest 6 columns are categorical type.
- There are null values in Transmission and Owner_type

Address Null value fields

- Transmission and Owner_type features have null values.
- Both the columns are categorical in nature.
- Applying Simple imputer(most_frequent) to remove null values.

```
from sklearn.impute import SimpleImputer
imp= SimpleImputer(strategy="most_frequent")

cars_df["Transmission"]= imp.fit_transform(cars_df["Transmission"].values.reshape(-1,1))
cars_df["Owner_Type"]= imp.fit_transform(cars_df["Owner_Type"].values.reshape(-1,1))
```

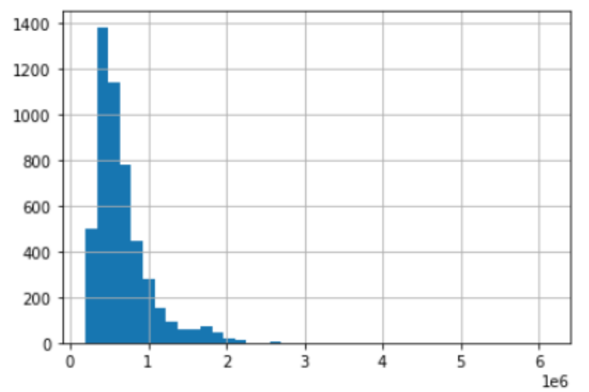
- Now there is no null values in the dataset.

Analyzing target Column-Selling Price

Before removing skewness

```
1 #histogram
2 cars_df['Selling Price'].hist(bins = 40)
```

<AxesSubplot:>

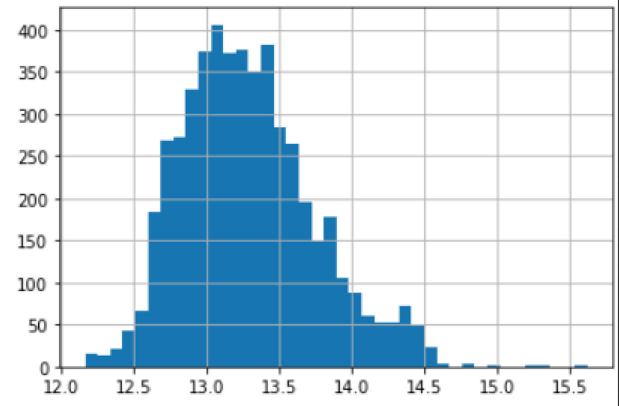



```
Skewness: 3.277941
Kurtosis: 25.861284
```

After removing skewness

```
5 cars_df['Selling Price'].hist(bins = 40)
```

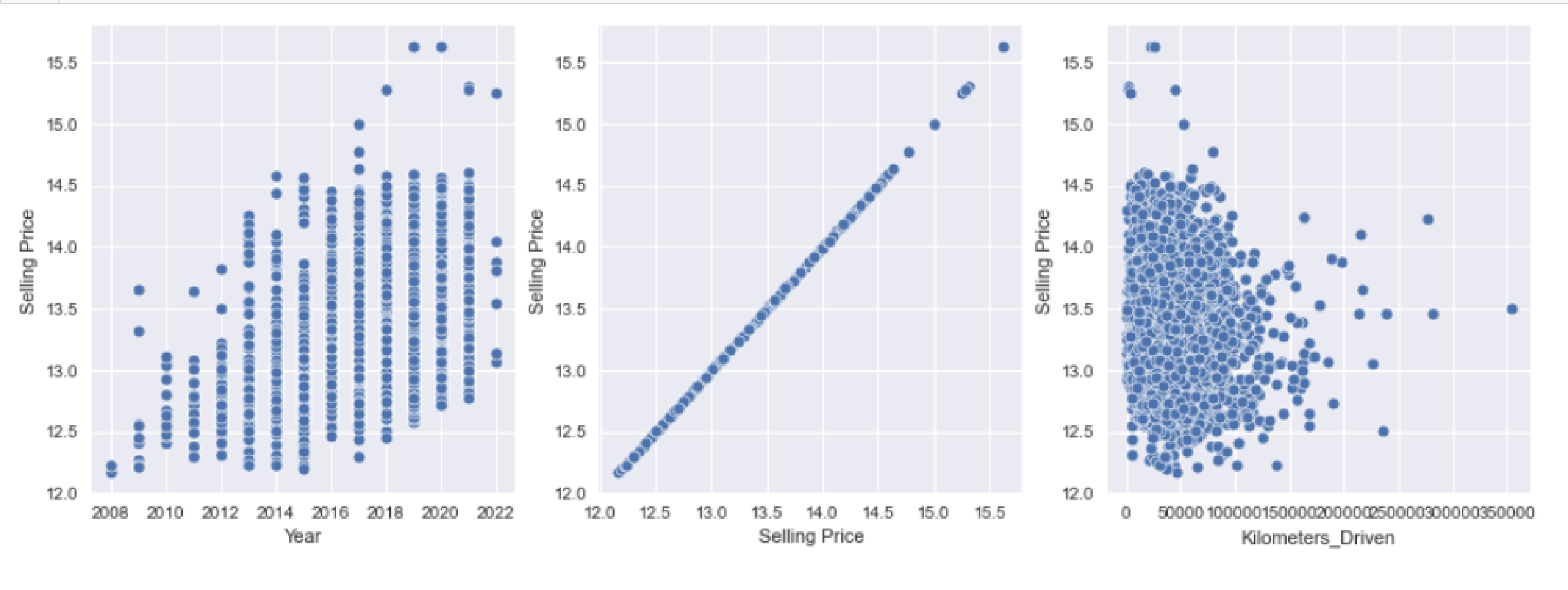
<AxesSubplot:>



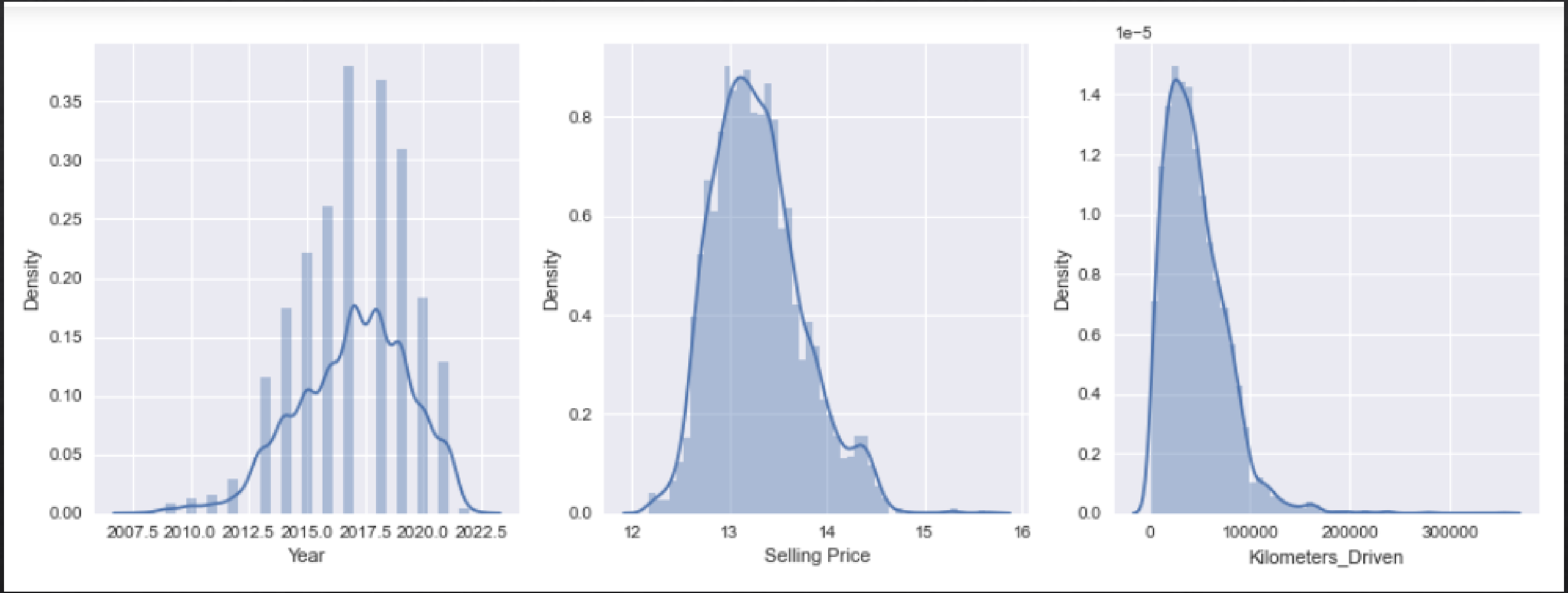

```
Skewness: 0.602862
Kurtosis: 0.395709
```

Since the target column is skewed towards right. So applying log transformation

Numerical Features analysis

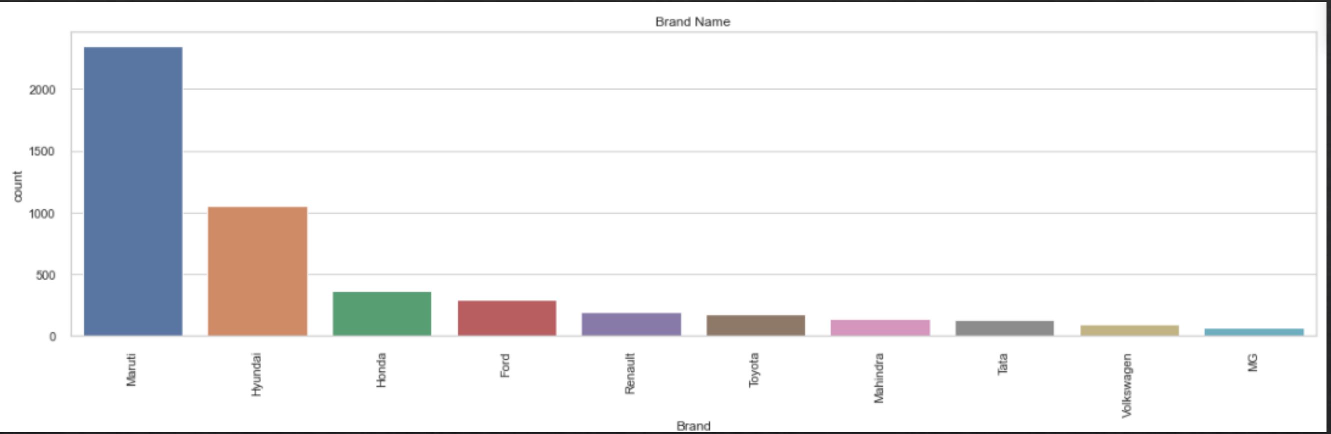


Numerical Features Analysis Cont..

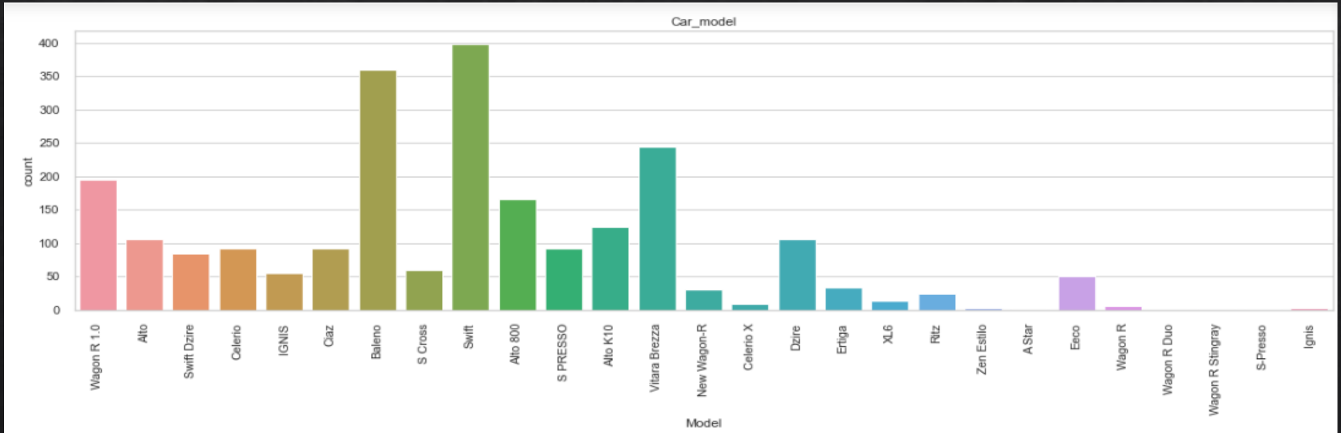


We can see the data in numerical columns are uniformly distributed..

Analyzing Categorical columns

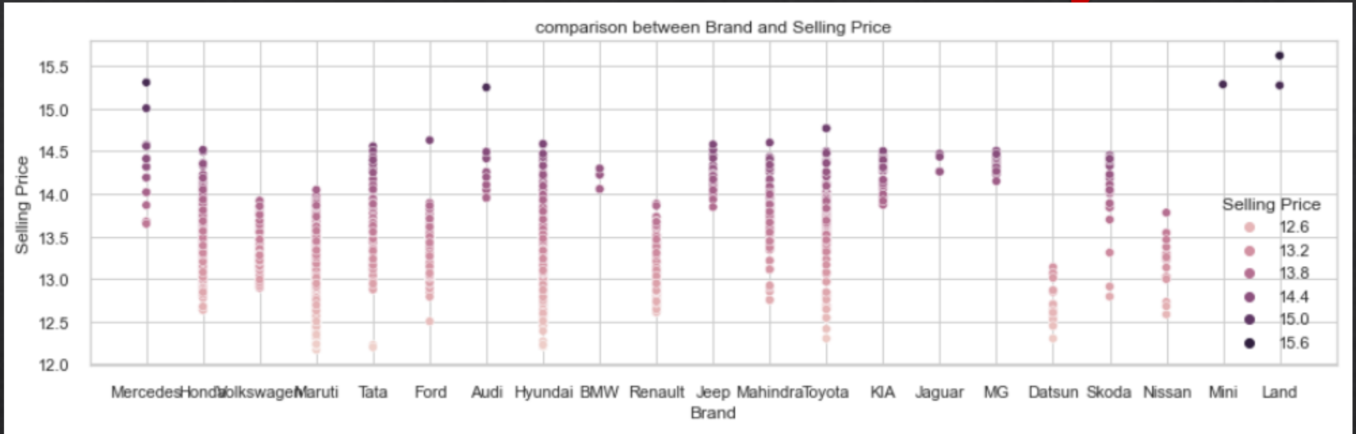


We can see from the given count plotfor brand , Maruti is the highest-selling brand followed by Hyundai. This may be because of their genuine price with genuine features

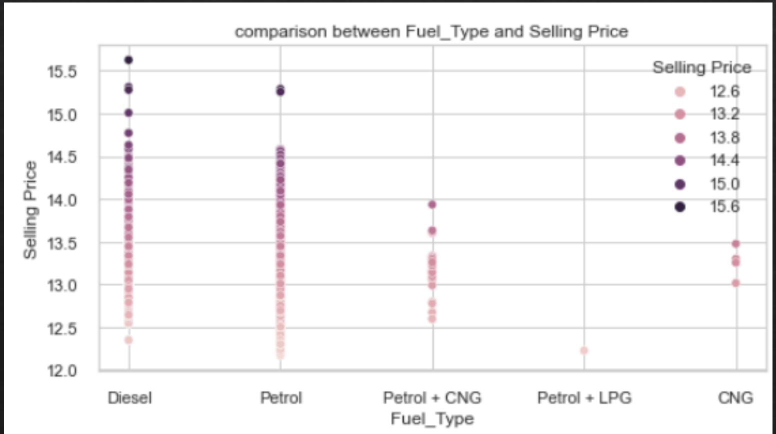


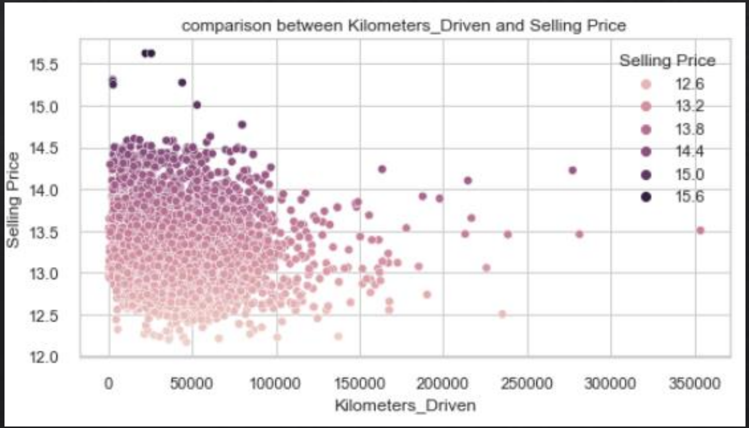
From the given count plot for model we can see Maruti Swift model is highest selling model, folled by Baleno And Brezza.

Bivariate analysis



Here we can Land Rover, Mercedes, Audi have highest selling Price as they are top brands





We can see from the above plot that Diesel cars are costly. They have the highest selling price, followed by petrol cars. Petrol+LPG have lowest selling price.

Selling Price does not depend much on Kilometer_driven. It is mainly affected by Model of the cars and the fuel type together with Kilometer Driven

Label Encoding

Converted all the categorical columns to numerical columns using Label encoder

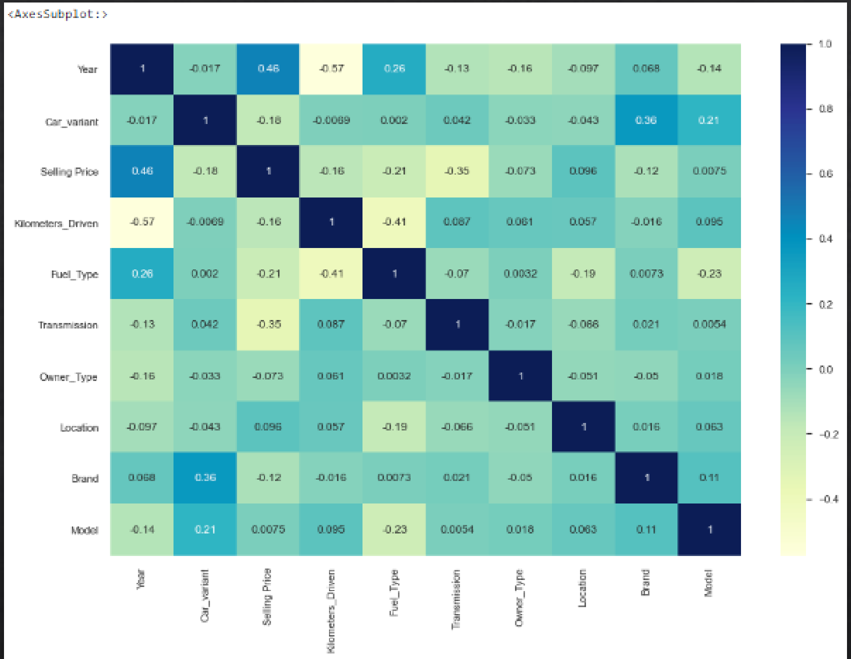
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
cars_df=cars_df.apply(LabelEncoder().fit_transform)
print(cars_df.head())
```

```
   Year  Car_variant  Selling Price  Kilometers_Driven  Fuel_Type  \
0     5          307           2779               2054          1
1     6          567           1396               2443          2
2     8          275            649               2693          2
3     6          390            246               2806          2
4    12          713           2780                320          2

   Transmission  Owner_Type  Location  Brand  Model
0             0           1         7     13     19
1             1           0         7      4     30
2             1           1         7     20     90
3             1           0         7     12    129
4             0           0         7      4     30
```

price and OverallQual, GarageCars And GarageArea,TotalBsmtSf and 1stFlrSF. But these columns have high correlation with target also

Checking the Multicollinearity

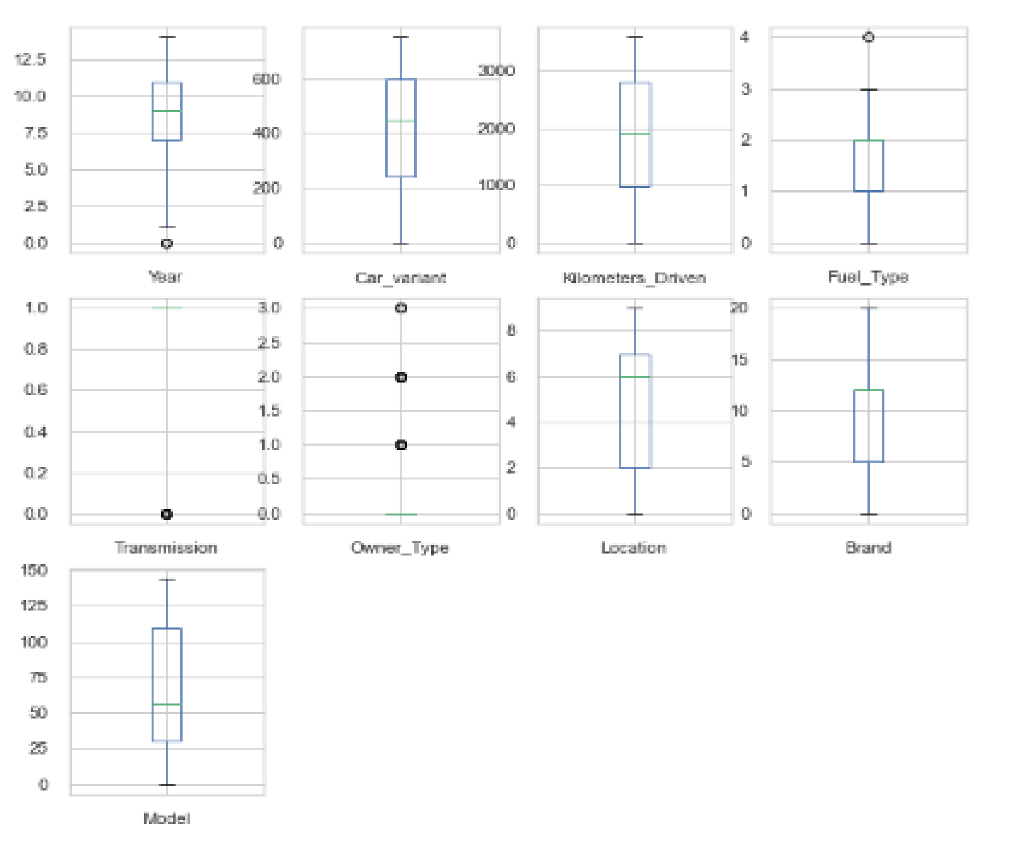


Observation:

- Year of manufacturing, Location, Brand, and Model is positively correlated with the selling price.
- It means more latest is the model and more recently the model was manufactured, more is the price.
- Other features are negatively correlated with the target feature.
- There is not much multicollinearity among the independent features.

Outliers check using Box Plot

There is not much outliers in numerical columns . Very Few are there in categorical columns which don't need to be removed.



Feature Scaling

```
X = cars_df.drop('Selling Price', axis=1)
y = cars_df['Selling Price'].values
```

By using a standard scaler, I have scaled the data in one range.

```
# Performing Standard scaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X
```

```
array([[-1.67527846, -0.52966998,  0.17577689, ...,  0.74026909,
         0.67237607, -1.11171284],
       [-1.25535882,  0.72876416,  0.54772628, ...,  0.74026909,
        -1.26827214, -0.85948336],
       [-0.41551954, -0.68455418,  0.7867683 , ...,  0.74026909,
         2.18176913,  0.51631381],
       ...,
       [ 1.26415903, -1.11532587,  1.21991245, ...,  0.74026909,
         1.96614155, -0.05693501],
       [ 0.00440011, -0.18118053,  1.32987178, ...,  0.74026909,
         0.45674849,  0.97491287],
       [ 0.84423939, -0.55871077, -0.15792577, ...,  0.74026909,
         0.45674849, -1.20343265]])
```

Model Building

Finding the best random state

```
maxScore = 0
maxRS = 0

for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=i)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    pred_train = lr.predict(x_train)
    pred_test = lr.predict(x_test)
    acc=r2_score(y_test,pred_test)
    if acc>maxScore:
        maxScore=acc
        maxRS=i
print('Best score is',maxScore,'on Random State',maxRS)
```

```
Best score is 0.5114352057819836 on Random State 65
```

Applying on 5 different algorithms

- LinearRegression(),
- Lasso()
- Ridge()
- DecisionTreeRegressor()
- KNeighborsRegressor()]

Train And test Scores of 5 Different Algorithms

```
Train Score of LinearRegression() is: 0.4457805727969435
r2_score 0.5114352057819836
mean_squred_error 362295.46902039496
RMSE 601.9098512405284

Train Score of Lasso() is: 0.4457639554424073
r2_score 0.5111749877628557
mean_squred_error 362488.4338234328
RMSE 602.0701236761652

Train Score of Ridge() is: 0.44578052283247993
r2_score 0.5114169961822433
mean_squred_error 362308.9723582463
RMSE 601.9210682126405

Train Score of DecisionTreeRegressor() is: 1.0
r2_score 0.9133786486041886
mean_squred_error 64234.10671936759
RMSE 253.44448449190523

Train Score of KNeighborsRegressor() is: 0.8265299936287829
r2_score 0.7436734225357128
mean_squred_error 190079.102513834
RMSE 435.9806217182525
```

- Have checked Multiple Model and their score also.
- I have found that Decision tree regressor model is overfitting. Other models are working well.
- But KNeighborsRegressor is having less train and test score difference with least mean square error and least RMSE.
- Now i will check with ensemble method to boost up score.

Ensemble Technique to boost up score

❖ Random Forest Regressor:

```
Train Score of RandomForestRegressor(random_state=65) is: 0.992142228456897
r2_score 0.9480528948680718
mean_squred_error 38521.401952727276
RMSE 196.26869835184436
```

❖ AdaBoostRegressor:

```
Train Score of AdaBoostRegressor(base_estimator=KNeighborsRegressor(), random_state=65) is: 0.91079138752551
r2_score 0.769914374515491
mean_squred_error 170620.11136758892
RMSE 413.06187353420665
```

❖ GradientBoostingRegressor:

```
Train Score of GradientBoostingRegressor() is: 0.8874196455796406
r2_score 0.8906350770059324
mean_squred_error 81099.61368365414
RMSE 284.77993904707216
```

Conclusion:

- Here we can see the least difference between train score and test score is coming in GradientBoostingRegressor.So the model is working well with the both train model and the test model.
- For AdaBoostRegressor, the difference is high, So the model is overfitting.
- For RandomForestRegressor, the difference is also more as compared to GradientBoostingRegressor.
- **So selecting GradientBoostingRegressor as final model**

Hyperparameter tuning to find best parameters of GradientBoost Regressor

```
#Using GradientBoostingRegressor for hyper parameter tuning
Gradient_Boost = GradientBoostingRegressor()
Para = {"n_estimators":[100,200,300,400],
        "learning_rate":[0.1,0.3,0.5],
        "max_depth" :[3,5,7,9,10]}

Rand_search = RandomizedSearchCV(Gradient_Boost,Para,cv = 6,scoring = "r2",n_jobs =-1,verbose = 2)
Rand_search.fit(X_train1,y_train1)
print(Rand_search.best_params_)
```

```
Fitting 6 folds for each of 10 candidates, totalling 60 fits
{'n_estimators': 400, 'max_depth': 5, 'learning_rate': 0.1}
```

```
prediction = Rand_search.predict(X_test1)
```

```
Selling_Price = GradientBoostingRegressor(n_estimators= 400, max_depth= 5, learning_rate =0.1)
Selling_Price.fit(x_train, y_train)
pred = Selling_Price.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(mean_squared_error(y_test, pred)))
```

```
R2_Score: 96.52564721843089
RMSE value: 154.02944685730992
```

Selecting Cv score as 6

Cross Validation

```
best_Gradient_Boost = GradientBoostingRegressor(n_estimators= 400, max_depth= 5, learning_rate =0.1)

for i in range(2,11):
    cross_score = cross_val_score(best_Gradient_Boost,X,y,cv = i,n_jobs = -1)
    print(i,"mean",cross_score.mean() ,"and STD" , cross_score.std())
```

```
2 mean 0.837907362057158 and STD 0.0027278780051092633
3 mean 0.9221170173995855 and STD 0.02685668813429838
4 mean 0.9284414753401908 and STD 0.03911204829957108
5 mean 0.9385125254718284 and STD 0.04225479783247203
6 mean 0.9469831137799503 and STD 0.04100712570982601
7 mean 0.9407102030673815 and STD 0.03980250225281623
8 mean 0.9476470703338274 and STD 0.04044537706039697
9 mean 0.9512029166911673 and STD 0.03808853477585622
10 mean 0.9474124136165762 and STD 0.041539243692988365
```

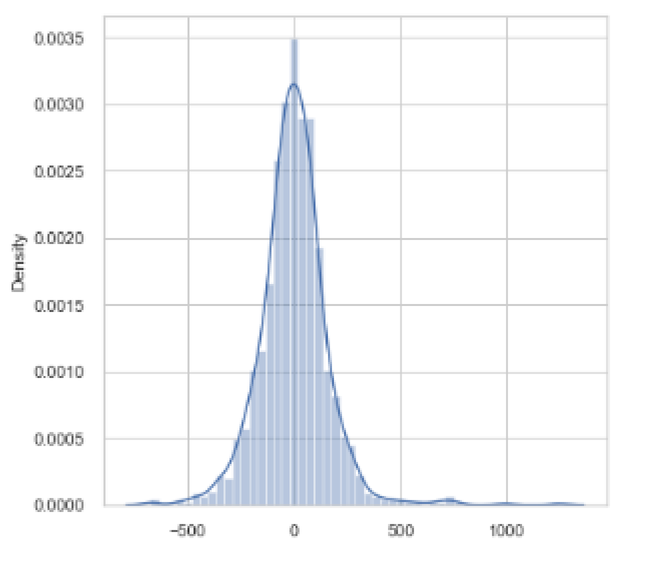
Selecting Cv score as 6

```
# Cross validate of RandomForestRegressor using cv=6
from sklearn.model_selection import cross_val_score
score=cross_val_score(best_Gradient_Boost,X,y,cv=6,scoring='r2')
print('Score:', score)
print('Mean Score:', score.mean())
print('Standard Deviation:', score.std())
```

```
Score: [0.99109706 0.98939463 0.96711154 0.93508075 0.92485076 0.87408461]
Mean Score: 0.94693655593532
Standard Deviation: 0.04101954186814009
```

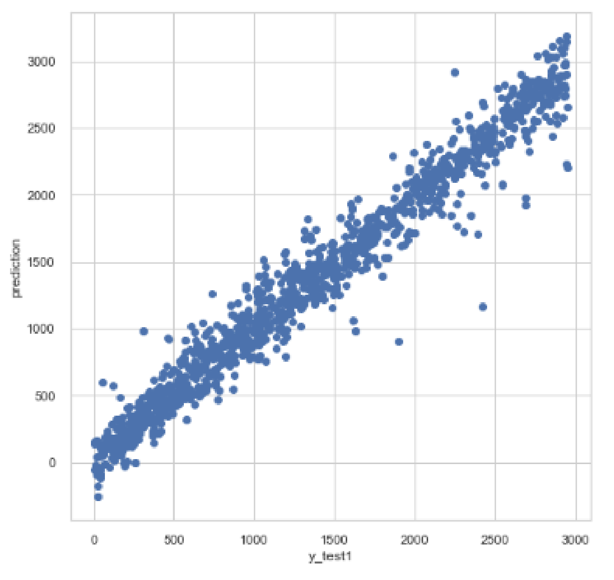
Plotting the residuals.

```
plt.figure(figsize = (6,6))
sns.distplot(y_test1-prediction)
plt.show()
```



Plotting y_test vs predictions.

```
plt.figure(figsize = (8,8))
plt.scatter(y_test1, prediction)
plt.xlabel("y_test1")
plt.ylabel("prediction")
plt.show()
```



Conclusion:

- After Scraping old car prices for cities like Delhi, Noida, Gurgaon, Mumbai, Pune, Hyderabad, Bangalore, Ahmedabad, Chennai, and Kolkata from different websites like Cars24 and CArdekho.com I have prepared an excel sheet and loaded the dataset for further EDA process.
- So, as we saw that we have done a complete EDA process, getting data insights, feature engineering, and data visualization as well so after all these steps one can go for the prediction using machine learning model-making steps.
- We have both numerical and categorical data types features in the datasets and the dependent variable of train data i.e. the Selling price is the numerical data type. So, I applied the regression method for prediction.
- Once data has been cleaned and missing value is replaced, Label encoding is applied to them to convert them into Numerical ones. I trained the model on five different algorithms but for most of the models, train and test data was having a variance, and the model was overfitting.
- Only Gradient Boost regressor worked well out of all the models, as there was less difference between train score and test score and RMSE was also low hence I used it as the final model and have done further processing.
- After applying hyperparameter tuning I got an accuracy(r2_score) of 96% from the GradientBoostRegressor model after hyper parameter tuning which is a good score.

Then I saved the model.

Limitations and Scope

- 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 the number of observations was only 5058. 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 cosmetic reconditioning time, used-to-new ratio, and appraisal-to-trade ratio.
- Another point that has room to improve is that the data cleaning process can be done more rigorously with the help of more technical information. For example, instead of using the ‘fill’ method, there might be indicators that help to fill missing values more meaningfully.

Thank You