**Capstone Project Report**

**Name :** Ritesh Sanjeevan Nikam

This report is for final capstone project submission.

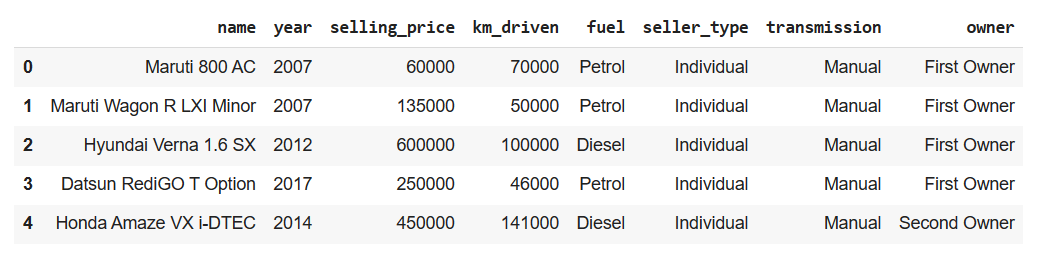
For this project we are provided with a dataset. First we will analyse this dataset and understand data. Following are the findings of this step.

**Summary of Dataset**

* name [Categorical] : name is combination of two data points brand + model.
* year [Numerical] : year columns represent the year in which the car is sold.
* selling price [Numerical] : this is our target variable. This feature is the selling price of the car.
* km\_driven [Numerical] : this total number of distances travelled by cars.
* fuel [Categorical] : this feature can have following values – Diesel, Petrol, CNG, PNG, Electric.
* seller\_type [Categorical] : this feature can have following values – Individual, Dealer, Trustmark Dealer.
* transmission [Categorical] : can be either manual or automatic.
* Owner [Categorical] : owner feature can take five values – First Owner, Second Owner, Third Owner, Fourth & Above Owner, Test Drive Car.

**Importing dataset as DataFrame.**

**df = pd.read\_csv('/content/drive/MyDrive/Capstone Project/CAR DETAILS.csv')**

****

After importing dataframe, we will perform data preprocessing in following steps.

**Step 1 :**  Handling null values.

**Step 2 :** Handling duplicate values.

**Step 3 :** Creating new feature ‘brand’ from ‘name’ feature.

**df['brand'] = df['name'].apply(lambda x : x.split(' ')[0])**

**df['model'] = df['name'].apply(lambda x : x.split(' ')[1:])**

**df.drop(['name', 'model'], axis = 1, inplace = True)**

**Step 4 :** Outlier handling on ‘km\_driven’ column.

**q1 = df['km\_driven'].quantile(0.25)**

**q3 = df['km\_driven'].quantile(0.75)**

**iqr = q3 - q1**

**min\_limit = q1 - 1.5 \* iqr**

**max\_limit = q3 + 1.5 \* iqr**

**df['km\_driven'] = np.where(df['km\_driven'] > max\_limit, max\_limit, df['km\_driven'])**

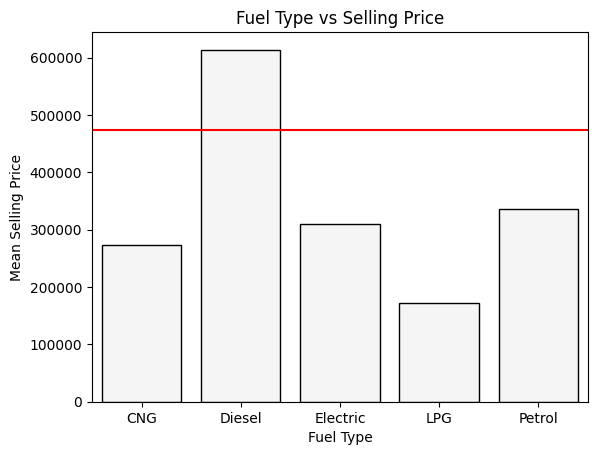
**df['km\_driven'] = np.where(df['km\_driven'] < min\_limit, min\_limit, df['km\_driven'])**

**Step 5 :** Analysing dataset we can see there are some categorical features on in the dataset

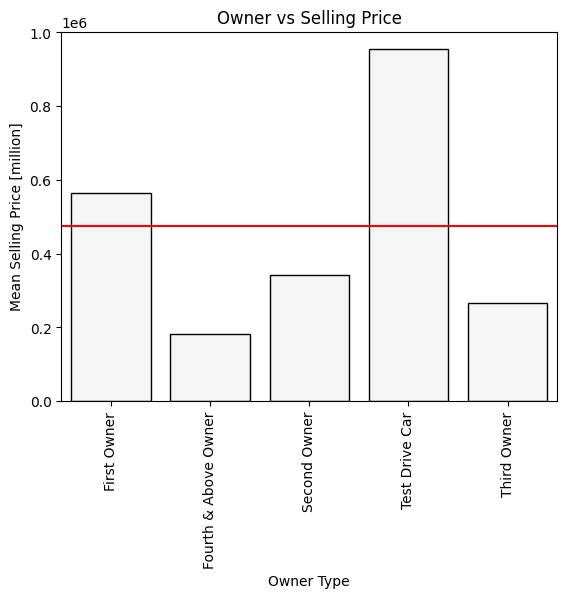
**Categorical columns :** brand, fuel, seller\_type, transmission, owner

We will use LabelEncoder on fuel, seller\_type, transmission and owner. And on brand features we will use OneHotEncoding.

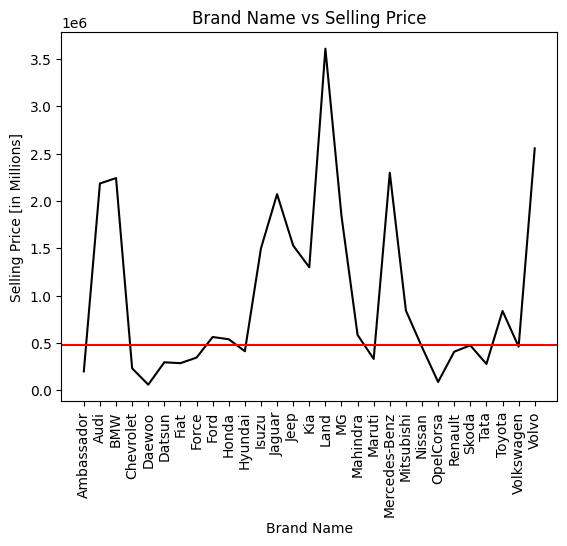
**Basic EDA**

****

We can observe that Cars powered by Diesel have higher selling prices than other car types.



Test drive cars have selling price higher than mean selling price.



We can see that there are certain brands that have selling price higher than other car brands.

**Separating Dependent and Independent Features**

**X = df\_encoded.drop('selling\_price', axis = 1)**

**y = df\_encoded['selling\_price']**

**Target variable is : selling\_price**

Selling price feature is a numeric feature so we can conclude that this is a regression **problem.**

We will apply the following regression models on the dataset one by one.

**Regression Models**

1. Linear regression.
2. Ridge regression.
3. Lasso regression
4. TreeRegressor.
5. Bagging
6. Gradient Boosting

Models are evaluated using the following functions.

**from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score**

**def model\_eval(y, y\_pred):**

**print(f'mean absolute error : {mean\_absolute\_error(y, y\_pred)}')**

**print(f'mean squared error : {mean\_squared\_error(y, y\_pred)}')**

**print(f'root mean squared error : {np.sqrt(mean\_squared\_error(y, y\_pred))}')**

**print(f'r2 score : {r2\_score(y, y\_pred)}')**

**def model\_score(model):**

**print(f'train score : {model.score(X\_train, y\_train)}')**

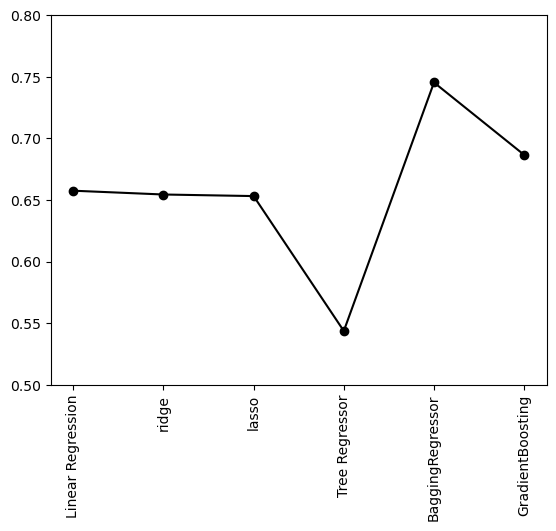
**print(f'test score : {model.score(X\_test, y\_test)}')**

**Generating Train and Test data.**

**from sklearn.model\_selection import train\_test\_splitX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 39)**

**Selecting Best Model**

Plotting all the models based on their r2\_score.



Following table has r2\_score for all the regressor models.

|  |  |
| --- | --- |
| **Linear Regression** | **0.66** |
| **Ridge Regression** | **0.65** |
| **Lasso Regression** | **0.65** |
| **Tree Regression** | **0.54** |
| **Bagging** | **0.74** |
| **Gradient Boosting Regressor** | **0.64** |

**Code snippet for Bagging Regressor**

Why use Bagging??

* Bagging minimizes the overfitting of data.
* It improves the model’s accuracy.
* It deals with higher dimensional data efficiently.

For base estimator for Bagging Regressor we will use our regressor model so far that is Gradient Boosting Regressor.

**Code Snippet**

**from sklearn.ensemble import BaggingRegressor**

**from sklearn.ensemble import GradientBoostingRegressor**

**from sklearn.model\_selection import GridSearchCV**

**gradient\_boosting\_reg = GradientBoostingRegressor(learning\_rate = 0.1, n\_estimators = 112, max\_depth=3)**

**model = BaggingRegressor()**

**parameter = {**

**'base\_estimator' : [gradient\_boosting\_reg],**

**'n\_estimators' : range(1, 50)**

**}**

**bagg\_reg\_cv = GridSearchCV(estimator = model, verbose = 2, param\_grid = parameter)**

**bagg\_reg\_cv.fit(X\_train, y\_train)**

**bagg\_reg\_cv.best\_params\_**

**Result**

{'base\_estimator': GradientBoostingRegressor(n\_estimators=112),

'n\_estimators': 11}

After hyperparameter tuning for n\_estimators we get

**N\_estimators = 11**

**bagg\_reg = BaggingRegressor(base\_estimator = gradient\_boosting\_reg, n\_estimators = 11)**

**bagg\_reg.fit(X\_train, y\_train)**

**y\_pred = bagg\_reg.predict(X\_test)**

**model\_eval(y\_test, y\_pred)**

**model\_score(bagg\_reg)**

**Result**

mean absolute error : 142952.72293419638

mean squared error : 67466457877.38185

root mean squared error : 259743.06126898146

r2 score : 0.74537009599182

train score : 0.8395869692932486

test score : 0.74537009599182