Capstone Project Report

Name : Ritesh Sanjeevan Nikam

This report is for final capstone project submission.

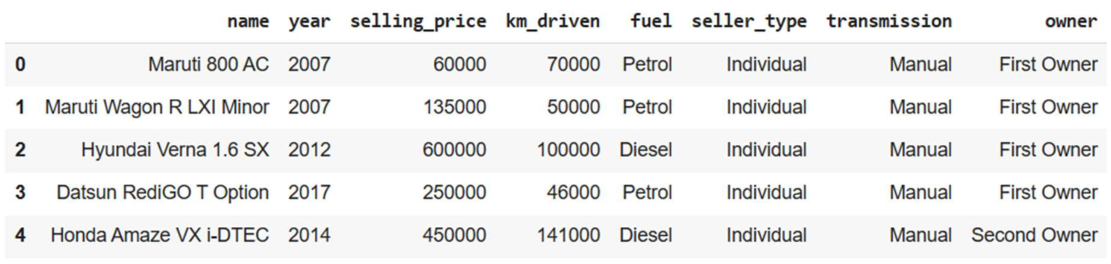
For this project we are provided with a dataset. First we will analyse this dataset and understand data. Following are the findings of this step.

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')
```



	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
1	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
2	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner
3	Datsun RediGO T Option	2017	250000	46000	Petrol	Individual	Manual	First Owner
4	Honda Amaze VX i-DTEC	2014	450000	141000	Diesel	Individual	Manual	Second Owner

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.

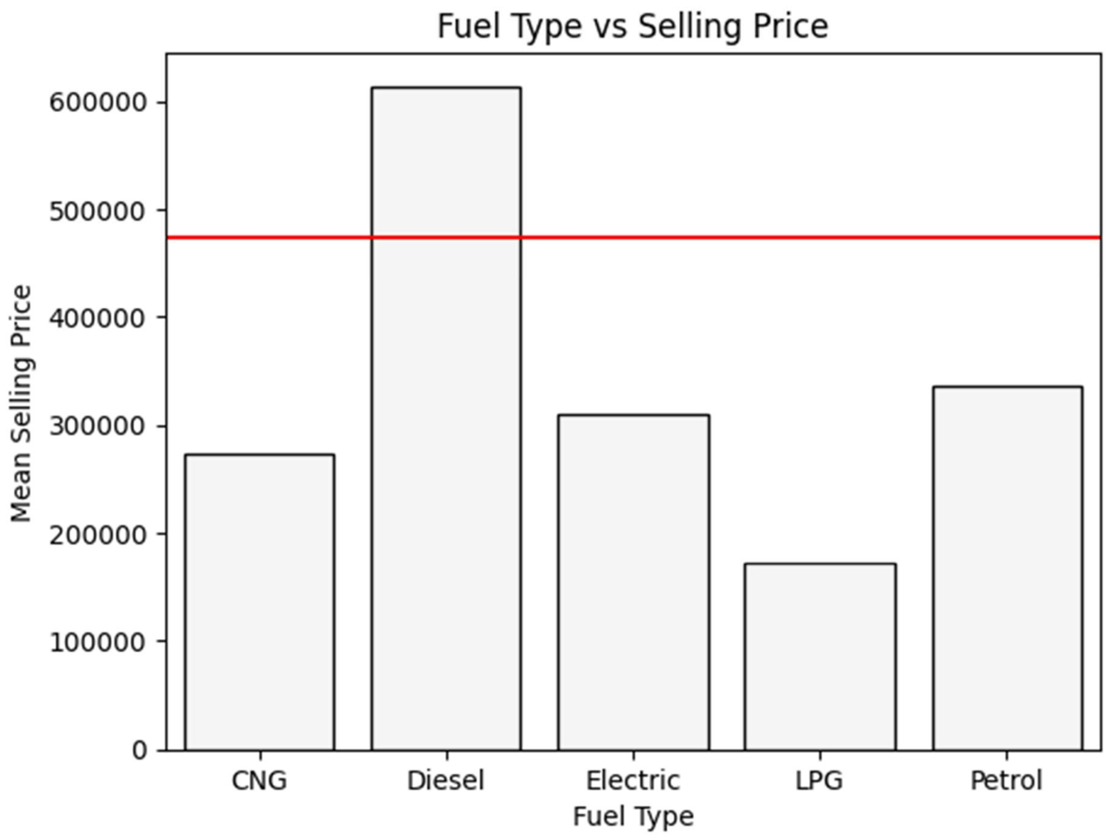
```
q1 = df['km_driven'].quantile(0.25)
q3 = df['km_driven'].quantile(0.75)
iqr = q3 - q1
min_limit = q1 - 1.5 * iqr
max_limit = q3 + 1.5 * iqr
df['km_driven'] = np.where(df['km_driven'] > max_limit, max_limit,
df['km_driven'])
df['km_driven'] = np.where(df['km_driven'] < min_limit, min_limit,
df['km_driven'])
```

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

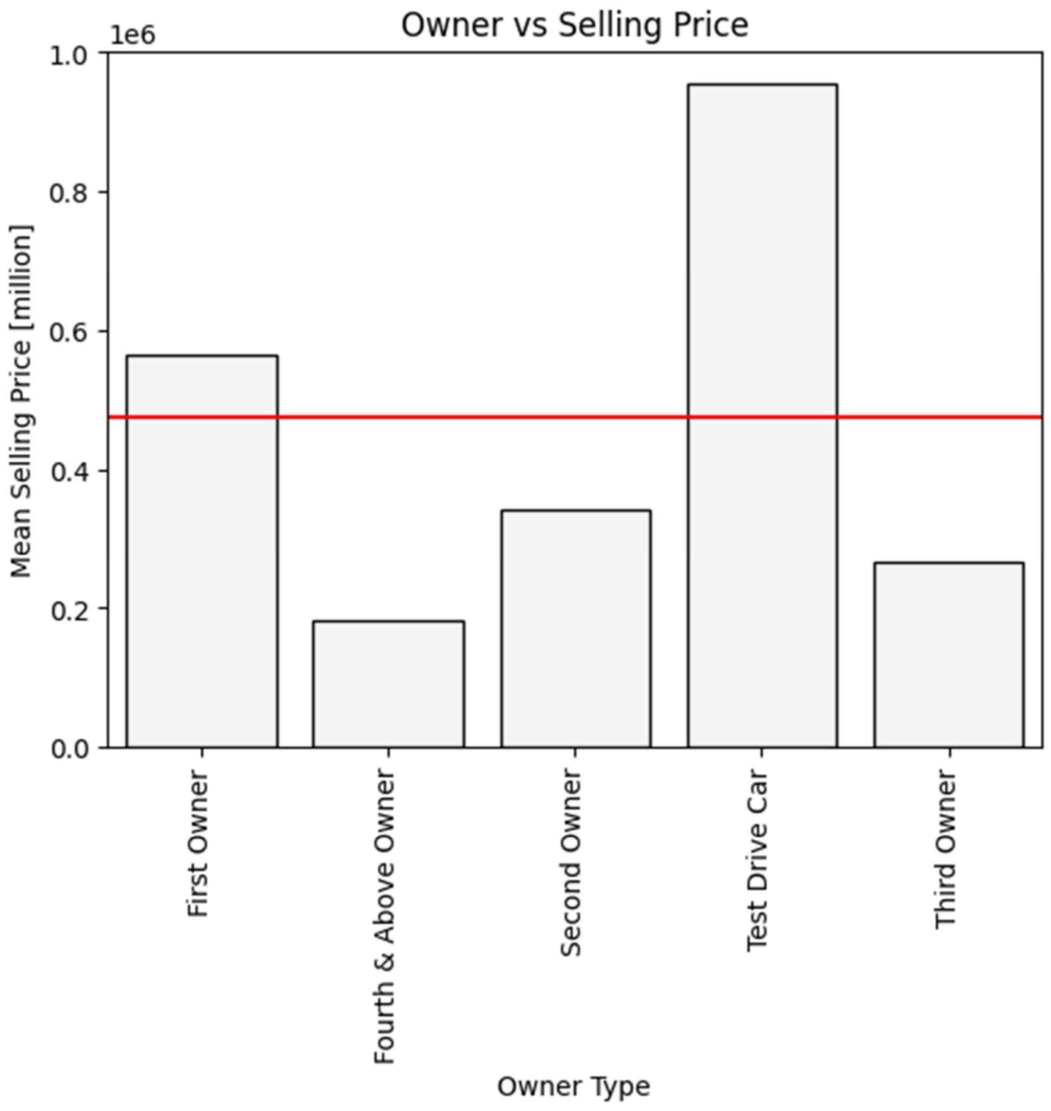
Categorical columns : brand, fuel, seller_type, transmission, owner

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

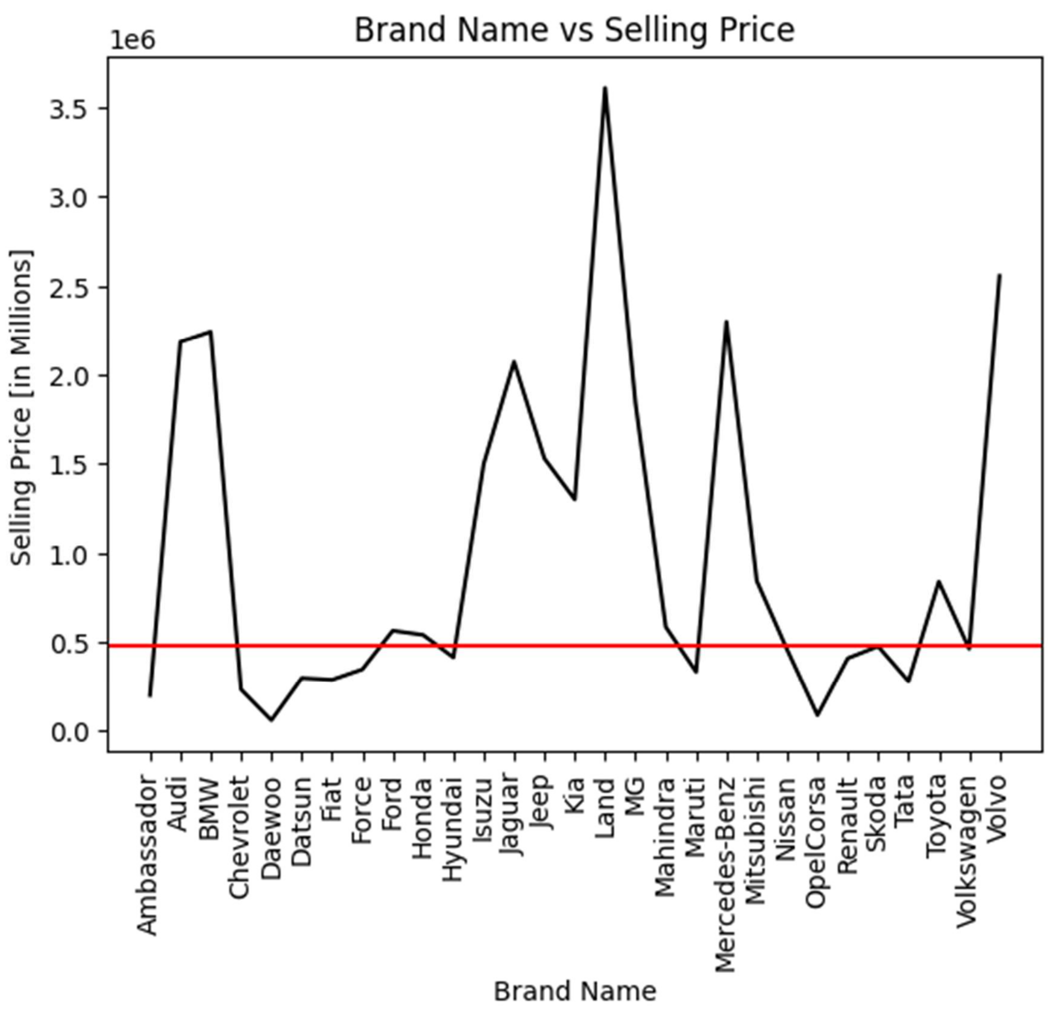
Basic EDA



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.

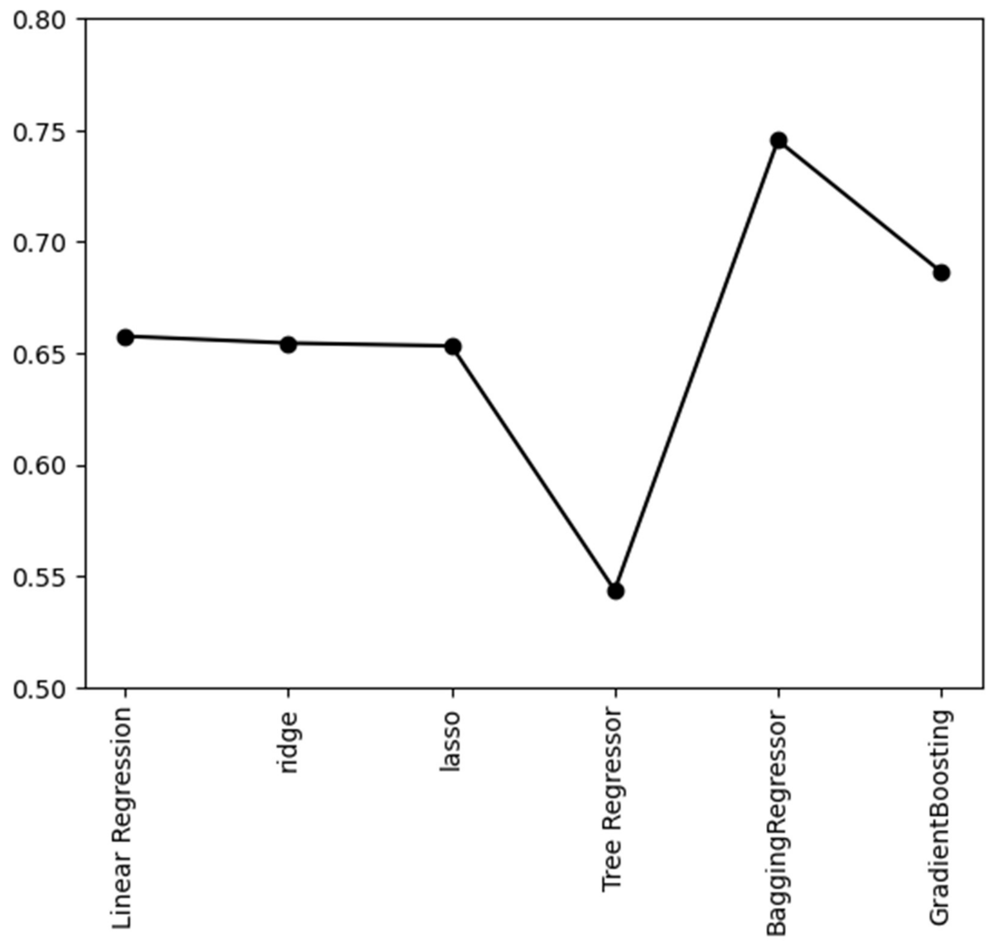
```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

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
```