Car Price Prediction Project

SUBMITTED BY:

AKANKSHA MISHRA

BATCH NO. 1840

ACKNOWLEDGEMENT

With the 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 hence cheaper.

The dataset has 6000 different kinds used cars data. The source of this data is www.olx.in.

The given dataset contains various Brands, Models, Kilometers driven, Manufacturing Year, Number of Owners, Fuel Type of the particular car, and finally the price of the car. These cars are selling in various locations in India. The given dataset includes all types of cars for example- SUV, Sedans, Coupe, etc.

INTRODUCTION

❖ BUSINESS PROBLEM FRAMING

Our client works with small traders, who sell used cars. With the change in market due to Covid-19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model.

ANALYTICAL PROBLEM FRAMING

❖ Mathematical/Analytical Modeling of the Problem:

We will begin with how the looks like in the Data frame, then we will be dealing with the Statistical summary of the data then we will look at the correlation between the various features with each other.

Description of the dataset:

Features:

The given dataset contains various Brands, Models, Kilometers driven, Manufacturing Year, Number of Owners, Fuel Type of the particular car, and finally the price of the car. These cars are selling in various locations in India. The given dataset includes all types of cars for example- SUV, Sedans, Coupe, etc.

Info of the dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Brand               6000 non-null   object
 1   Year                6000 non-null   object
 2   Kilometers Driven   6000 non-null   object
 3   Fuel Type           6000 non-null   object
 4   No of Owners        6000 non-null   object
 5   Price(in rupees)    6000 non-null   object
dtypes: object(6)
memory usage: 281.4+ KB
```

The above image gives the idea about the data types of the given data in the dataset. From above, all the data has the "Object" data type and no column in the dataset has the null values as the Non-Null Count represents the "non-null" value.

The names of the columns are "Brand", "Year", "Kilometers Driven", "Fuel Type", "No of Owners", and "Price (in rupees)".

Null Values:

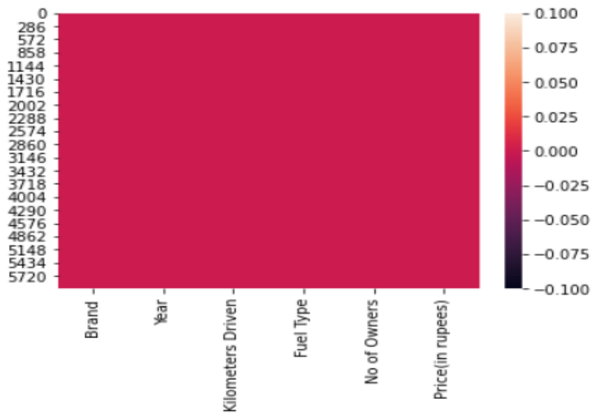
We have no null values in the dataset.

```
# Missing Values
print(df.isna().sum())
```

```
Brand                 0
Year                  0
Kilometers Driven     0
Fuel Type             0
No of Owners          0
Price(in rupees)      0
dtype: int64
```

```
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



The heatmap above represents the Null values present in the dataset.

Statistical Summary:

It gives the basic statistics about the data like the percentile, mean, maximum, minimum etc.

Statistical Summary: ¶

```
df.describe()
```

	Brand	Year	Kilometers Driven	Fuel Type	No of Owners	Price(in rupees)
count	6000.000000	6000.000000	6000.000000	6000.000000	6000.000000	6000.000000
mean	15.393333	9.962000	20.939667	2.479000	2.115000	18.730833
std	9.598592	4.921454	12.598551	0.851673	0.958433	11.856959
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	8.000000	7.000000	11.000000	2.000000	2.000000	9.000000
50%	15.000000	11.000000	22.000000	3.000000	2.000000	20.000000
75%	24.000000	13.000000	32.000000	3.000000	3.000000	29.000000
max	33.000000	18.000000	41.000000	3.000000	5.000000	39.000000

Correlation:

Correlation:

```
df.corr()
```

	Brand	Year	Kilometers Driven	Fuel Type	No of Owners	Price(in rupees)
Brand	1.000000	0.204771	0.147224	0.442662	0.230403	0.338158
Year	0.204771	1.000000	0.168779	0.263584	0.189154	0.479037
Kilometers Driven	0.147224	0.168779	1.000000	0.229312	0.680626	0.413661
Fuel Type	0.442662	0.263584	0.229312	1.000000	0.420578	0.207588
No of Owners	0.230403	0.189154	0.680626	0.420578	1.000000	0.341083
Price(in rupees)	0.338158	0.479037	0.413661	0.207588	0.341083	1.000000

❖ Data Sources and their formats

I have extracted the data from www.olx.in website. This dataset includes total 6000 cars which belongs various brands, models, different manufacturing years, and selling in the different parts of India.

❖ Data Pre-processing Done

The column “No of Owners” has repeat count of owners so I have merged it into the respective category of the owner.
As well as the “Kilometers Driven” column has the commas in the values so I need to remove that.

```
df['No of Owners'].unique()

array(['1st', 'Second', '4th', '2nd', '3rd', 'First', '--', ' '],
      dtype=object)

# Merging the 'First' and '1st owners' to 1st owners.
# Merging the 'Second' and '2nd owners' to 2nd owners.

df['No of Owners'] = df['No of Owners'].replace(['First'],'1st')

df['No of Owners'] = df['No of Owners'].replace(['Second'],'2nd')

df['No of Owners'].value_counts()

1st     3314
2nd     1370
--       598
         417
3rd      151
4th      150
Name: No of Owners, dtype: int64
```

```
# removing the commas in kilometers value.
df['Kilometers Driven'] = df['Kilometers Driven'].str.replace(r',', '')
```

```
df['Kilometers Driven'].unique
```

```
<bound method Series.unique of 0       56000.0
1       38000.0
2       68716.0
3        999999
4         90000
         ...
5995    93141.0
5996    63000.0
5997    55000.0
5998    62000.0
5999    37000.0
Name: Kilometers Driven, Length: 6000, dtype: object>
```

MODEL DEVELOPMENT AND EVALUATION

❖ Identification of possible problem-solving approaches (methods)

Spliting the data into Feature and Target:

```
x = df.drop(columns = "Price(in rupees)")
y = df["Price(in rupees)"]
```

```
x.shape
```

```
(6000, 5)
```

```
y.shape
```

```
(6000,)
```

We will be split the data into target and feature as x and y respectively.

Scalling and the x and y

Scalling:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

Getting the best accuracy score and a specific random state

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=i)
    lr.fit(x_train,y_train)
    pred=lr.predict(x_test)
    acc=r2_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("We are getting the Best Accuracy is",maxAccu," on Random_state",maxRS)
```

```
We are getting the Best Accuracy is 0.43630550971551596  on Random_state 177
```

Model Building:

Train Test Split the data:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=177)
```

```
x_train.shape
```

```
(4200, 5)
```

```
x_test.shape
```

```
(1800, 5)
```

```
y_train.shape
```

```
(4200,)
```

```
y_test.shape
```

```
(1800,)
```

```
# Linear Regression

lr=LinearRegression()
lr.fit(x_train,y_train)
pred=lr.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 43.6305509715516%
```

```
dtr=DecisionTreeRegressor()
dtr.fit(x_train,y_train)
pred=dtr.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 100.0%
```

```
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
pred=rfr.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 99.99881383600024%
```

```
ls=Lasso()
ls.fit(x_train,y_train)
pred=ls.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 40.445995780776755%
```

```
knn=KNeighborsRegressor()
knn.fit(x_train,y_train)
pred=knn.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 99.92532460439786%
```

```
gbr=GradientBoostingRegressor()
gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 99.8135798160431%
```

```
svr=SVR()
svr.fit(x_train,y_train)
pred=svr.predict(x_test)
acc=r2_score(y_test,pred)

# Getting the accuarcy score
print(f"R2 Score: {acc*100}%")
```

```
R2 Score: 87.91083955045978%
```

Cross Validation Score:

Cross Validation:

```
cvlr=cross_val_score(lr,x,y,cv=5).mean()
print("Cross Validation Score for Linear Regression is : ",cvlr)
```

```
Cross Validation Score for Linear Regression is :  0.3884759812772919
```

```
cvdtr=cross_val_score(dtr,x,y,cv=5).mean()
print("Cross Validation Score for Decision Tree Regressor is : ",cvdtr)
```

```
Cross Validation Score for Decision Tree Regressor is :  0.9689112050235437
```

```
cvrfr=cross_val_score(rfr,x,y,cv=5).mean()
print("Cross Validation Score for Random Forest Regressorr is : ",cvrfr)
```

```
Cross Validation Score for Random Forest Regressorr is :  0.9695755591096742
```

```
cvls=cross_val_score(ls,x,y,cv=5).mean()
print("Cross Validation Score for Lasso is : ",cvls)
```

```
Cross Validation Score for Lasso is :  0.3687863162837826
```

```
cvknn=cross_val_score(knn,x,y,cv=5).mean()
print("Cross Validation Score for KNeighborsRegressor is : ",cvknn)
```

```
Cross Validation Score for KNeighborsRegressor is :  0.9782412198789132
```

```
cvgbr=cross_val_score(gbr,x,y,cv=5).mean()
print("Cross Validation Score for Gradient Boosting Regressor is : ",cvgbr)
```

```
Cross Validation Score for Gradient Boosting Regressor is :  0.9867439710209449
```

```
cvsvr=cross_val_score(svr,x,y,cv=5).mean()
print("Cross Validation Score for SVR is : ",cvsvr)
```

```
Cross Validation Score for SVR is :  0.8566645057955331
```

Gradient Boosting Regressor:

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Hyper Parameter Tuning of the model:

Hyper Parameter Tuning:

We are selecting Gradient Boosting Regressor as our best model as it has least difference between it's Accuracy score and CV score.

```
from sklearn.model_selection import GridSearchCV
```

```
parameter={'max_depth':[100,200,500,1000],
           'n_estimators':[10,100,500],
          'subsample':[0.001,0.01,0.1,1.0],
           'random_state':[50,100,200]}
```

```
GCV = GridSearchCV(GradientBoostingRegressor(),parameter,cv=5)
```

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(),
             param_grid={'max_depth': [100, 200, 500, 1000],
                         'n_estimators': [10, 100, 500],
                         'random_state': [50, 100, 200],
                         'subsample': [0.001, 0.01, 0.1, 1.0]})
```

```
''' Getting the best parameters using GridSearchCV '''

GCV.best_params_
```

```
{'max_depth': 100, 'n_estimators': 500, 'random_state': 50, 'subsample': 1.0}
```

```
car_price_final=GradientBoostingRegressor(max_depth=100, n_estimators=500, random_state=50, subsample=1.0)
car_price_final.fit(x_train,y_train)
pred=car_price_final.predict(x_test)
acc=r2_score(y_test,pred)
print(acc*100)
```

```
100.0
```

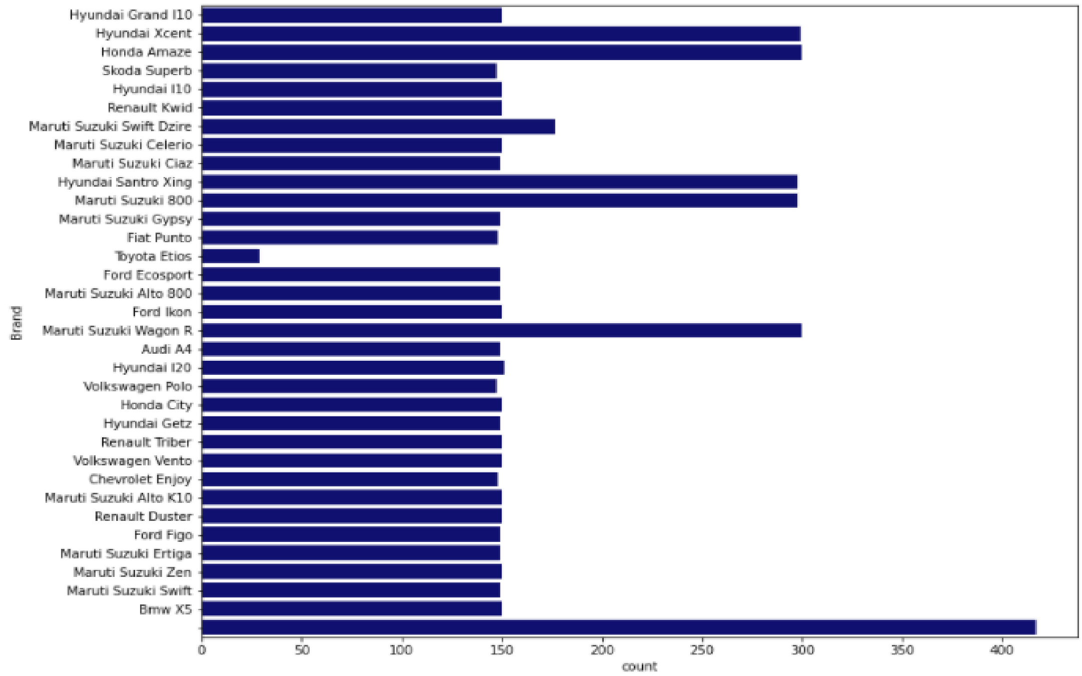
We are getting the model accuracy and cross validation score both as 100.0% which shows our model is performing well.

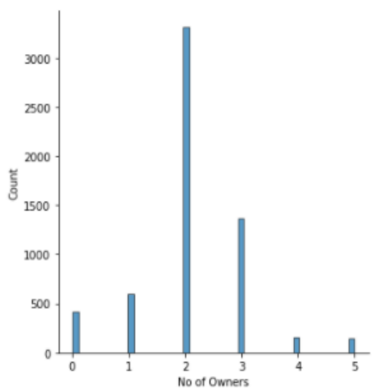
Here we are getting our Model Accuracy Score and Cross Validation Score both as 100.0%.

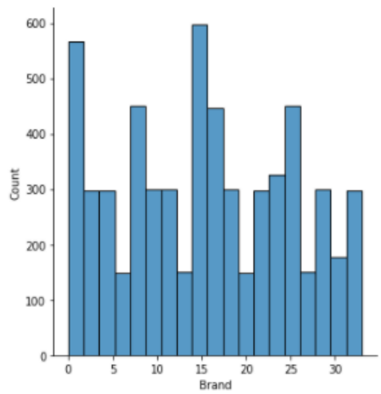
❖ Visualization

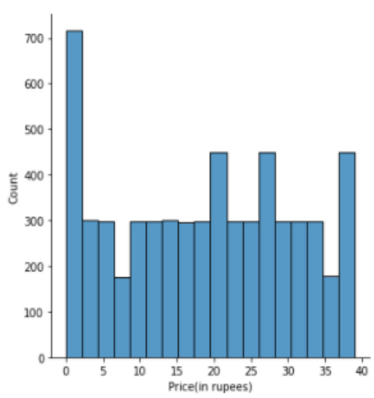
Visualization:

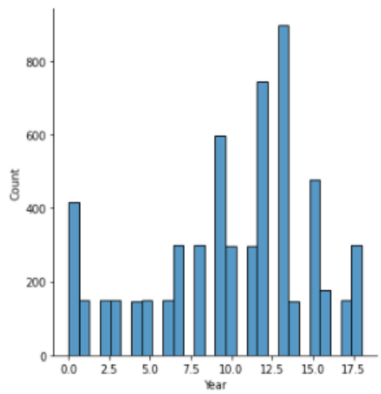
```
plt.figure(figsize=(12,10))
sns.countplot(y="Brand", data=df, color="navy")
plt.show()
```





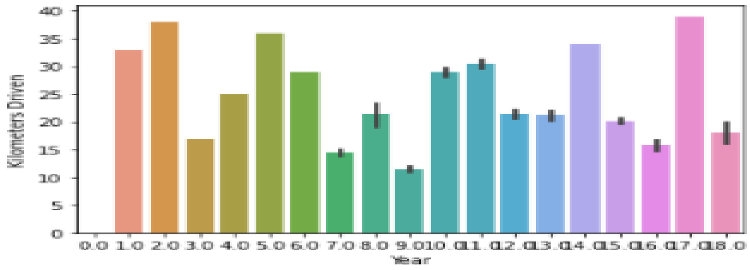






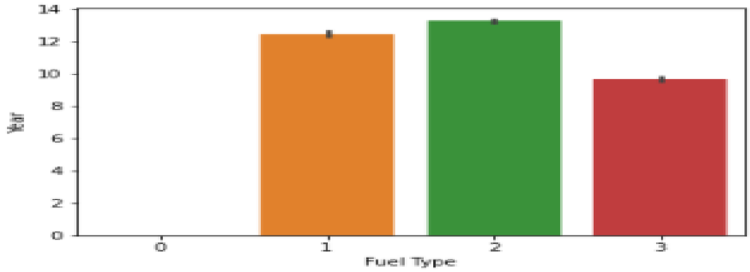
```
sns.barplot('Year','Kilometers Driven', data = df)
```

<AxesSubplot:xlabel='Year', ylabel='Kilometers Driven'>

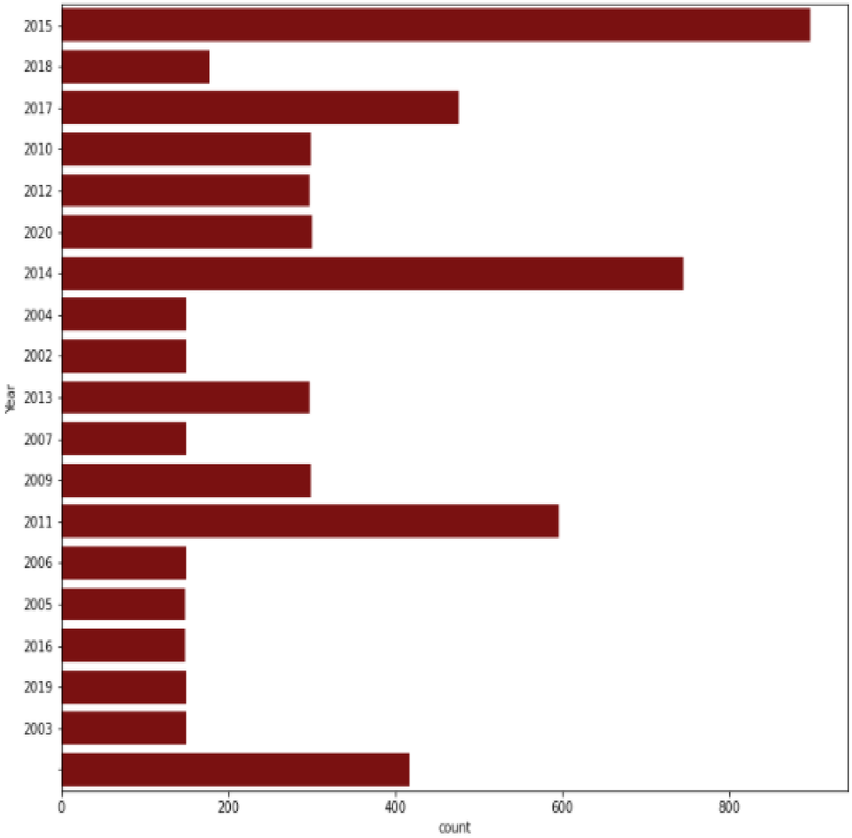



```
sns.barplot('Fuel Type','Year', data = df)
```

<AxesSubplot:xlabel='Fuel Type', ylabel='Year'>



```
plt.figure(figsize=(12,10))
sns.countplot(y="Year", data=df, color="darkred")
plt.show()
```



CONCLUSION

❖ Conclusion:

Saving the model:

```
import joblib
joblib.dump(car_price_final,"Car_price_prediction(submission).pkl")
```

```
['Car_price_prediction(submission).pkl']
```

```
Car_price_model = joblib.load(open('Car_price_prediction(submission).pkl','rb'))
result = Car_price_model.score(x_test,y_test)
print(result)
```

```
1.0
```

```
Conclusion = pd.DataFrame([Car_price_model.predict(x_test)[:],gbr.predict(x_test)[:]],index=["Predicted","Original"])
Conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	1790	1791	1792	
Predicted	15.000000	12.000000	4.000000	2.477026e-22	24.000000	28.000000	31.000000	3.00000	19.000000	19.000000	...	33.000000	2.477026e-22	37.000000	1
Original	15.052928	12.380229	5.111348	-8.604932e-02	23.912117	27.497938	31.150626	3.61037	18.970422	18.970422	...	32.658343	-8.604932e-02	36.785034	1

2 rows × 1800 columns

Our model is showing the best accuracy and cv score as 100.0%. Hence, we can conclude that our model is performing best.

----- --:-- ----- ----- --:-- ----- ❄ ----- --:-- ---------- --:-- -----