FLIP ROBO

CAR PRICE PREDICTION MODEL

Submitted by:

BHAVNA PIPLANI

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped and guided me in completion of the project.

INTRODUCTION

Problem Framing:

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. One of our clients 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.

Description of the domain related concepts that will be useful for better understanding of the project.

Data Cleaning, Data Visualisation using different plotting methods like barplot, countplot, wordcloud,distplot and, Data pre-processing using labelEncoder and pipelining for Model Training

B. **Analytical Problem Framing**

a) Mathematical/ Analytical Modelling of the Problem

Statistical modelling is the process of applying statistical analysis to a dataset. A statistical model is a mathematical representation (or mathematical model) of observed data.

When data analysts apply various statistical models to the data they are investigating, they are able to understand and interpret the information more strategically. Rather than sifting through the raw data, this practice allows them to identify relationships between variables, make predictions about future sets of data, and visualize that data so that non-analysts and stakeholders can consume and leverage it.

most common techniques will fall into the following two groups:

Supervised learning, including regression and classification models.

Unsupervised learning, including clustering algorithms and association rules.

b) **Data Sources and their formats**

Data Set Description

The data set contains 6099 samples. All the data samples contain 6 fields which includes columns- name, year, km_driven,fuel,transmission, Price of the car

C. Explanatory Data Analysis

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter("ignore")
```

Importing the dataset from the source file provided.

```
#importing dataset
df=pd.read_csv("newCarDetails.csv",sep='\t',usecols=['name','year','se
df
```

	name	year	selling_price	km_driven	fuel	transmission
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Manual
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Manual
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Manual
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Manual
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Manual
...	...	...	...	...	...	...
6094	Maruti Swift Dzire VDI	2015	615000	60000	Diesel	Manual
6095	Maruti Ertiga ZDI	2014	684000	60000	Diesel	Manual
6096	Maruti Ertiga ZDI	2014	680000	64000	Diesel	Manual

After importing the dataset, checking the datatypes of each column.

```
1  df.dtypes #checking the datatypes of variables
```

```
name              object
year               int64
selling_price      int64
km_driven          int64
fuel              object
transmission      object
dtype: object
```

Train dataset has 6099 rows and 6 columns

```
1  df.shape # checking no of columns and rows
```

```
(6099, 6)
```

After checking the no. of rows and columns count, we will check the null values if any, column count, datatypes of columns.

```python
df.isnull().sum() # checking the null values
```

```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
transmission     0
dtype: int64
```

The dataset is pretty clean. No Data is missing in any of the features. There are no null values in the dataset.

After checking for null values , we will see the values count for each class .

```python
# checking for the count of unique values in each column
for i in df.columns:
    print("Count of unique values of  ", i, "is " , df[i].nunique())
```

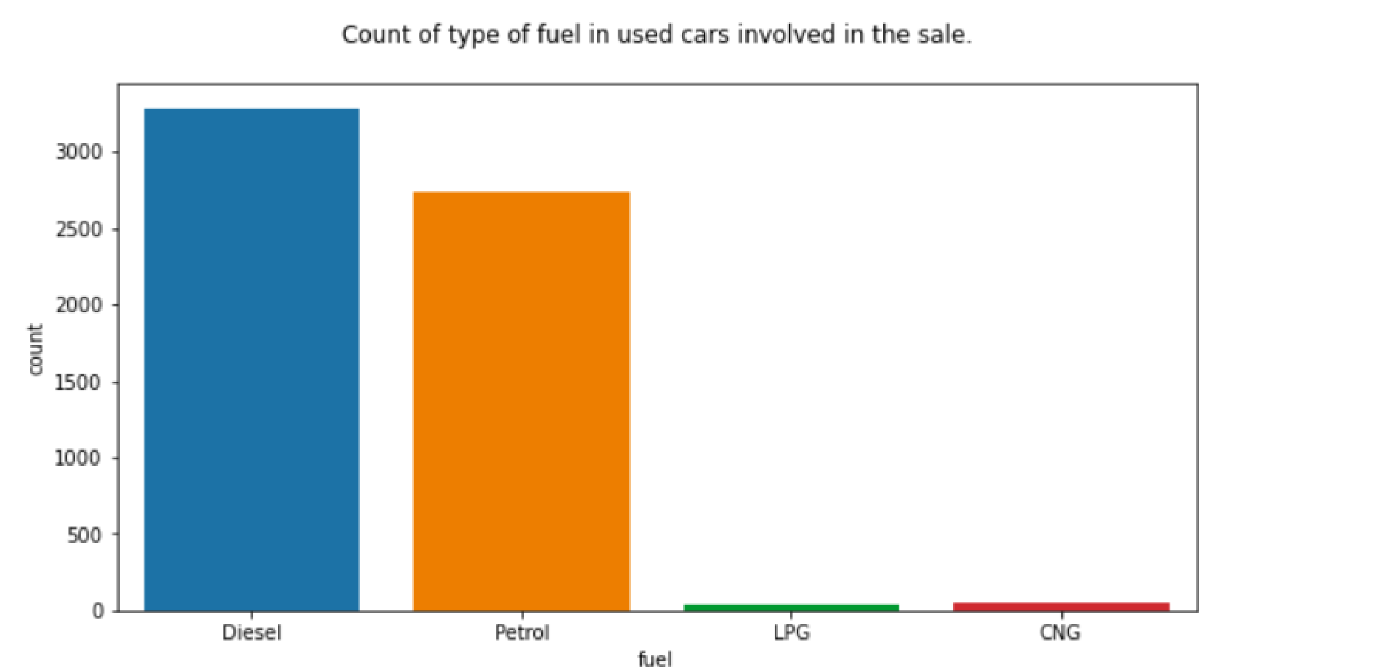
```
Count of unique values of   name is  1802
Count of unique values of   year is  29
Count of unique values of   selling_price is  603
Count of unique values of   km_driven is  749
Count of unique values of   fuel is  4
Count of unique values of   transmission is  2
```

D. Data Preprocessing

No spacing, special characters, numerical text etc are there in the dataset. Its pretty clean. After visualization, we will do the label encoding for categorical variables. We can go ahead with the data visualization.

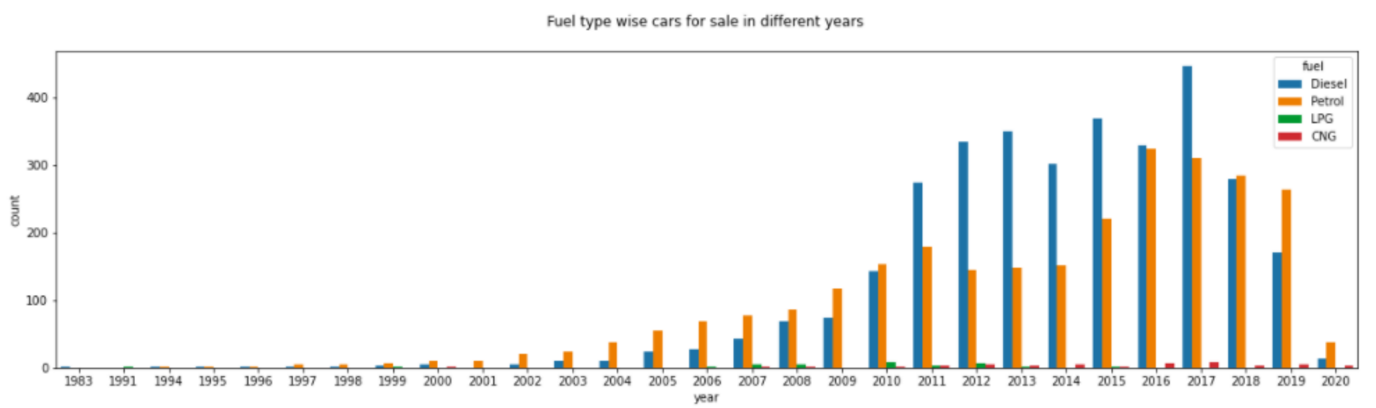
Data Visualization

```
plt.figure(figsize=(10,5))
sns.countplot(df["fuel"])
plt.title("Count of type of fuel in used cars involved in the sale.\n ")
plt.show()
```



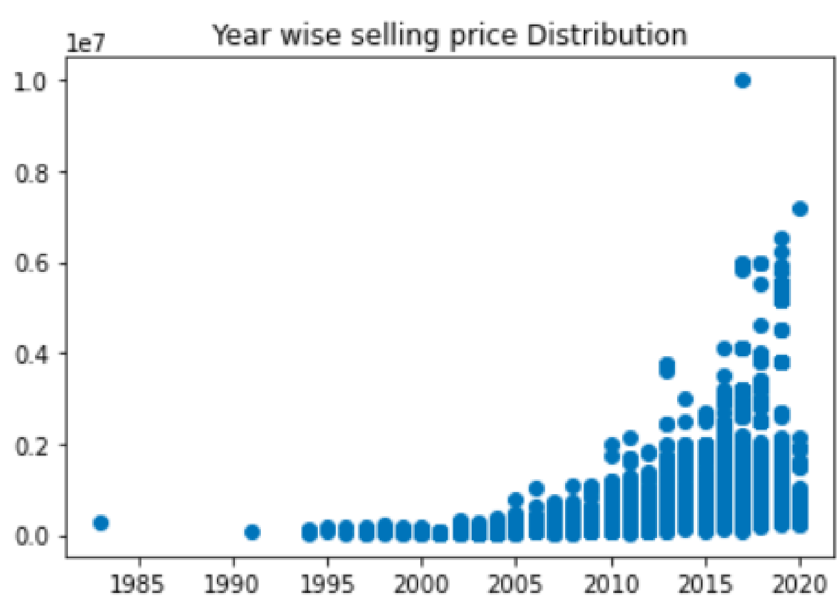
Above countplot shows that maximum no of diesel fuel cars are in sale. CNG & LPG cars are the least ones for sale.

```python
plt.figure(figsize=(20,5))
sns.countplot(x ='year',hue='fuel', data = df)
plt.title('Fuel type wise cars for sale in different years \n ')
plt.show()
```



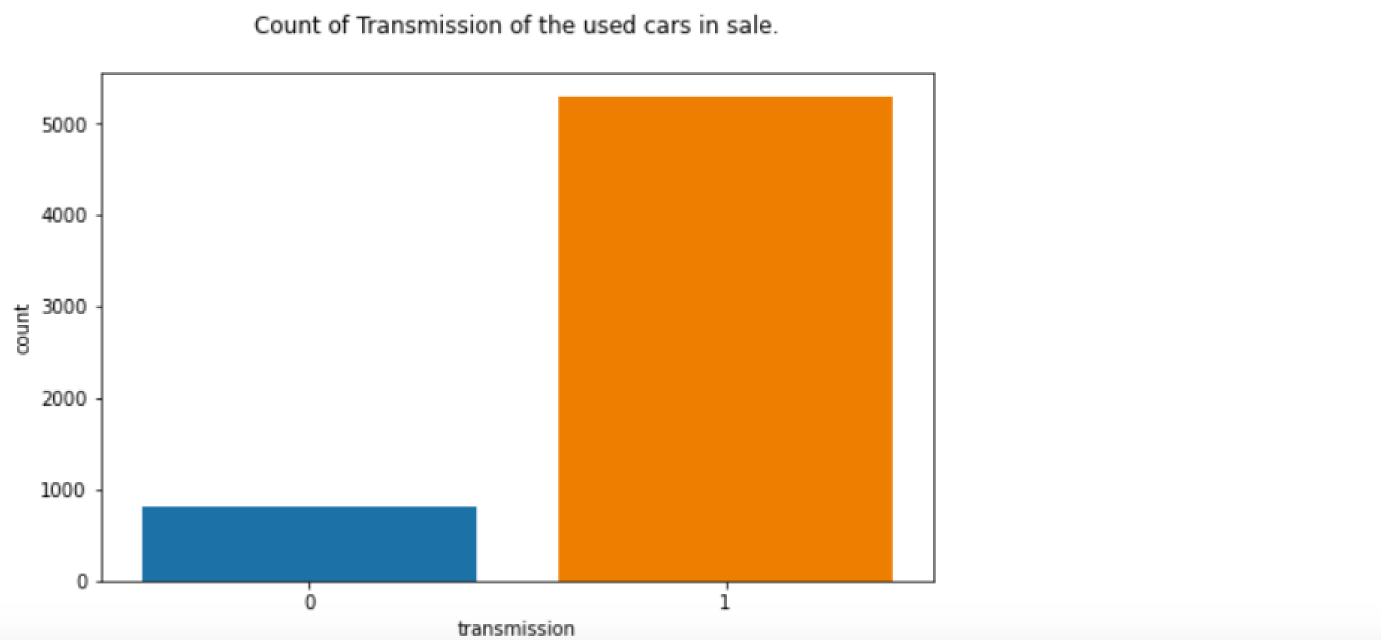
Above countplot shows that highest no of diesel cars were at sale in year 2017 and petrol cars in 2016.

```python
#plt.figure(figsize=(10,5))
plt.title('Year wise selling price Distribution')
plt.scatter(df['year'],df['selling_price'])
plt.show()
```



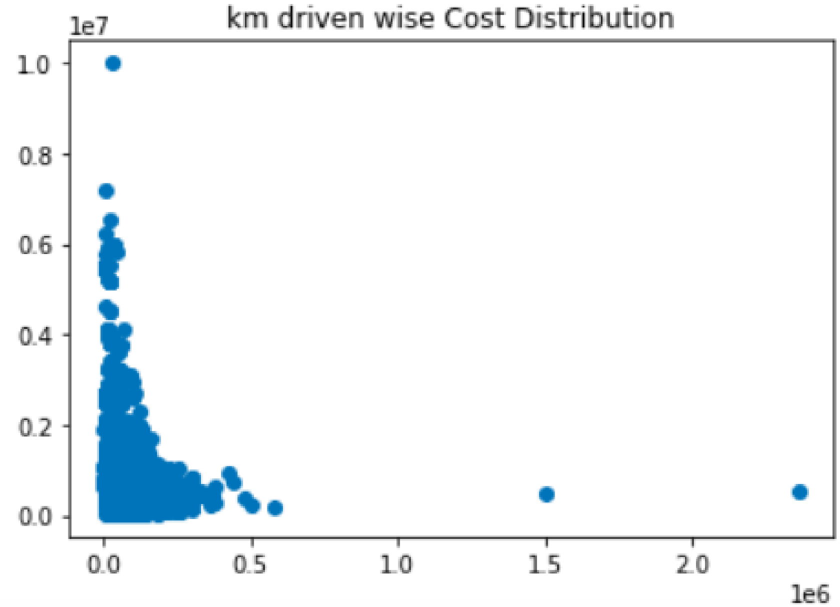
Above scatterplot shows increase in selling price after the year 2015.

```
plt.figure(figsize=(8,5))
sns.countplot(df["transmission"])
plt.title("Count of Transmission of the used cars in sale.\n ")
plt.show()
```



Above countplot shows manually transmitted used cars are maximum in sale.

```
plt.title('km driven wise Cost Distribution')
plt.scatter(df['km_driven'],df['selling_price'])
plt.show()
```



After visualizing few of variable correlation, lets go ahead with LabelEncoding to change the nominal values to numeric ones to further proceed for training the model.

Data preprocessing - Label Encoding

```python
# extracting all the categorical columns
df_cat=df.select_dtypes(include=['object']).columns.tolist()
df_cat
```

```
['name', 'fuel', 'transmission']
```

```python
# changing the nominal value to integer for training model
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
list1=['name', 'fuel', 'transmission']
for val in df_cat:
    df[val]=le.fit_transform(df[val].astype(str))
```

Visualization to show the correlation between variables.

```python
plt.figure(figsize=(10,5)) # visualization for correlation
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
```

```
<AxesSubplot:>
```

key observations from the heatmap:-

1. transmission and seller type are negatively correlated with target variable.

2. year and selling price are highly correlated.

Now, lets check for skewness in data

```
df.skew()# checking for skewness
```

```
name              -0.057822
year              -1.069080
selling_price      4.181611
km_driven         12.706186
fuel               0.164965
transmission      -2.177153
dtype: float64
```

Above image shows data is highly skewed . After splitting the X and y variables , we will remove skewness.

```
# seperatng the target variable - Price
df_x=df.drop(columns=['selling_price'])
y_t=pd.DataFrame(df['selling_price'])
print(df_x.shape, y_t.shape)
```

```
(6099, 5) (6099, 1)
```

```
from sklearn.preprocessing import power_transform
```

```
df_x=power_transform(df_x,method='yeo-johnson') # removing the skewness
```

After removing the skewness , we will do data scaling using StandardScalar.

```
#scaling the dataset
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaledX=sc.fit_transform(df_x)
scaledX.shape
```

```
(6099, 5)
```

Lets go ahead with model training.

Libraries and packages used for model training are listed below

Data Modelling

```
# importing our libraries
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor, ExtraTreesRegressor
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, r2_score,mean_absolute_error,mean_squared_error
```

E. Model Development and Evaluation

Identification of possible problem-solving approaches (methods)most common techniques will fall into the following two groups:

Supervised learning, including regression and classification models.

Unsupervised learning, including clustering algorithms and association rules.

For this dataset, I will be using classification model because the output variable is nominal.

Testing of Identified Approaches (Algorithms)

Here, In this project I will be using LinearRegression(), BaggingRegressor(), ExtraTreesRegressor(), DecisionTreeRegressor(),RandomForestRegressor(), GradientBoostingRegressor()algorithms

3. Model Training and metrics representation

Lets train our model and search for best accuracy on best random state.

finding the best random state

```
best_rstate=0
accuracy=0
for i in range(30,200):
    x_train,x_test,y_train,y_test=train_test_split(scaledX,y_t,test_size=.22,random_state=i
    mod=RandomForestRegressor()
    mod.fit(x_train,y_train)
    predlr=mod.predict(x_test)
    tempaccu=r2_score(y_test,predlr)
    if(tempaccu>accuracy):
        accuracy=tempaccu
        best_rstate=i

print("Best Accuracy",accuracy*100, "Random state",best_rstate)
```

```
Best Accuracy 96.56932504286156 Random state 121
```

```
x_train,x_test,y_train,y_test=train_test_split(scaledX,y_t,test_size=.22,random_state=121)
```

```
x_train.shape , x_test.shape
```

```
((4757, 5), (1342, 5))
```

```
y_train.shape , y_test.shape
```

```
((4757, 1), (1342, 1))
```

After train test split , we will train the model

```
#using algorithms in for loops
model=[LinearRegression(),BaggingRegressor(),ExtraTreesRegressor(),DecisionTreeRegressor(),
for m in model:
    m.fit(x_train,y_train)
    y_pred=m.predict(x_test)
    r2score=r2_score(y_test,y_pred)
    cvscore=cross_val_score(m,x_train,y_train,cv=5).mean()
    print(m , "\nAccuracy Score of " ,r2score*100, "Cross Val Score", {cvscore*100})
    print("**********************************************************************\n")
```

```
LinearRegression()
Accuracy Score of  50.39707298042586 Cross Val Score {47.43097839899371}
**********************************************************************

BaggingRegressor()
Accuracy Score of  96.04688377468011 Cross Val Score {90.06583258561365}
**********************************************************************

ExtraTreesRegressor()
Accuracy Score of  94.62853158960407 Cross Val Score {89.44247593671201}
**********************************************************************

DecisionTreeRegressor()
Accuracy Score of  94.92543512960394 Cross Val Score {87.9033091707932}
**********************************************************************

RandomForestRegressor()
Accuracy Score of  96.60378446920109 Cross Val Score {90.76873280717501}
**********************************************************************
```

Lets do the hyperparameter tuning with best performing model i.e RandomForestRegressor using GridSearchCV

Hyperparameter Tuning

```
# GradientBoostingRegressor is best performing model so finding its best parameter
from sklearn.model_selection import GridSearchCV
```

```
# Create the parameter grid based on the results of random search
param_grid = {'bootstrap':[True],'criterion':['mse'] ,
              'max_features':['auto'],  'bootstrap': [True], 'max_depth': [58, 60, None], '
```

```
rfr = RandomForestRegressor()# performing GridSearchCV with parameters
g_search = GridSearchCV(estimator = rfr, param_grid = param_grid,
                        cv = 5, n_jobs = -1)
```

```
g_search.fit(x_train, y_train);# passing parametrs to train model
print(g_search.best_params_) # finding best parameters
```

```
{'bootstrap': True, 'criterion': 'mse', 'max_depth': 58, 'max_features': 'auto', 'n_estimator
s': 29}
```

Lets train our best model again with best parameters

```
# implementing with best parameters
rf = RandomForestRegressor(bootstrap= True, criterion ='mse', max_depth =58, max_features=
rf = rf.fit(x_train, y_train)
print(" Score is ",rf.score(x_train,y_train))
predrf = rf.predict(x_test)
print("Mean Absolute error  " , mean_absolute_error(y_test,predrf))
print("Mean Squared error \n",mean_squared_error(y_test,predrf))
print("Root mean Squared error is \n",np.sqrt(mean_squared_error(y_test,predrf)))
print("r2 score " , r2_score(y_test,predrf))
print("**********************************************************************\n")
```

```
 Score is  0.9853078244543971
Mean Absolute error   81431.1849945534
Mean Squared error
 24284559949.593544
Root mean Squared error is
 155835.04082713087
r2 score  0.9624839376711817
**********************************************************************
```

RandomForestRegressor() is best performing model with r2score 96.24% and cross validation score with 90.76%.

Lets save our model.

Saving the model- Serialization

```python
# saving the prediction model

import pickle
filename="carprice.pkl"
pickle.dump(rf,open(filename,'wb'))
```

```python
pred=rf.predict(x_test)
```

```python
ds_pred=pd.DataFrame(data=pred.round(2),columns=['price'])
ds_pred
```

	price
0	614482.76
1	138620.69
2	391758.62
3	645000.00
4	100499.97

CONCLUSION

Key Findings and Conclusions of the Study

1. Dataset was pretty clean. Required no data cleaning.
2. transmission and seller_type are negatively correlated with target variable.
3. year and selling price are highly correlated.
4. r2_score,mean_absolute_error, mean_squared_error metrics were used.
5. RandomForestRegressor was the best fit model.