# Project_1_Vehicle_price_detection

February 27, 2025

[1]:
```
#loading required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#loading dataset
df = pd.read_csv('dataset.csv')
df.columns
```

[2]:
```
Index(['name', 'description', 'make', 'model', 'year', 'price', 'engine',
       'cylinders', 'fuel', 'mileage', 'transmission', 'trim', 'body', 'doors',
       'exterior_color', 'interior_color', 'drivetrain'],
      dtype='object')
```

[3]:
```
#remove irrelavant columns
df.drop(columns=['name', 'description'], inplace=True)
```

[4]:
```
#checking null values
df.isnull().sum(axis=0)
```

[4]:
```
make               0
model              0
year               0
price             23
engine             2
cylinders        105
fuel               7
mileage           34
transmission       2
trim               1
body               3
doors              7
exterior_color     5
interior_color    38
drivetrain         0
```

```
dtype: int64
```

[5]:
```
df.shape
```

[5]:
```
(1002, 15)
```

[6]:
```
#drop missing values in target variable
df = df.dropna(subset='price')
```

[7]:
```
df.shape
```

[7]:
```
(979, 15)
```

[8]:
```
#filling mileage with median
df['mileage'].fillna(df['mileage'].median(), inplace=True)

#filling cylinder and doors with mode
df['cylinders'].fillna(df['cylinders'].mode()[0], inplace=True)
df['doors'].fillna(df['doors'].mode()[0], inplace=True)

df.isnull().sum(axis=0)
```

[8]:
```
make              0
model             0
year              0
price             0
engine            2
cylinders         0
fuel              7
mileage           0
transmission      2
trim              1
body              3
doors             0
exterior_color    5
interior_color   37
drivetrain        0
dtype: int64
```

[9]:
```
#filling Null values for categorical columns using mode

cat_col = ['make', 'model', 'engine', 'fuel','transmission',
           'trim', 'body', 'exterior_color', 'interior_color', 'drivetrain']

for col in cat_col:
    df[col].fillna(df[col].mode()[0], inplace=True)

df.isnull().sum(axis=0)
```

[9]:
```
make              0
model             0
year              0
price             0
engine            0
cylinders         0
fuel              0
mileage           0
transmission      0
trim              0
body              0
doors             0
exterior_color    0
interior_color    0
drivetrain        0
dtype: int64
```

missing value done.

[10]:
```
#seperate feature and target
X = df.drop(columns='price')
y = df[['price']]
```

[11]:
```
#check numerial columns
numerical_col = X.select_dtypes(include=np.number).columns.tolist()
numerical_col
```

[11]:
```
['year', 'cylinders', 'mileage', 'doors']
```

[12]:
```
#check categorical columns
categorical_col = X.select_dtypes(include='object').columns.tolist()
categorical_col
```

[12]:
```
['make',
 'model',
 'engine',
 'fuel',
 'transmission',
 'trim',
 'body',
 'exterior_color',
 'interior_color',
 'drivetrain']
```

[13]:
```
#check no. of unique value are present in each col.
for col in categorical_col:
    print(f'{col}: {len(df[col].unique())}')
```

```
make: 28
```

```
model: 151
engine: 100
fuel: 7
transmission: 38
trim: 197
body: 8
exterior_color: 262
interior_color: 90
drivetrain: 4
```

**low cardinality Nominal : unique variable in column <= 40**

**high cardinality Nominal: unique variable in column > 40**

**if columns have more unique variable then we will consider target encoding instead of one-hot-encoder.**

[14]:
```
#selecting columns for one - hot encoding (low cardinality Nominal)
onehot_cols = ['make', 'fuel', "transmission", 'body', 'drivetrain']

#select columns for target encoding (high cardinality Nominal)
target_cols = ['model', 'engine', 'trim', 'exterior_color', 'interior_color']
```

[15]:
```
onehot_cols
```

[15]:
```
['make', 'fuel', 'transmission', 'body', 'drivetrain']
```

[16]:
```
from sklearn.preprocessing import OneHotEncoder
```

[17]:
```
from category_encoders import TargetEncoder
```

[18]:
```
#apply onehot_encoding
onehot_encoder = OneHotEncoder(drop='first', sparse_output=False)
onehot_encoder.fit(X[onehot_cols])

encoded_col = onehot_encoder.transform(X[onehot_cols])
encoded_df = pd.DataFrame(encoded_col, columns=onehot_encoder.
 ↪get_feature_names_out(onehot_cols))
```

[19]:
```
encoded_df.shape
```

[19]:
```
(979, 80)
```

[20]:
```
#apply target encoder for categorical columns have more no. of unique variables.
target_encoder = TargetEncoder(cols=target_cols)
target_encoder.fit(X[target_cols], y['price'])
target_encoded_col = target_encoder.transform(X[target_cols])
target_encoded_col.reset_index(drop=True, inplace=True) #reset index
```

[21]:
```
#drop those columns which have converted using encoder ex. onehot encoder and␣
 ↪target encoder
X.drop(columns=['make', 'fuel', "transmission", 'body', 'drivetrain',
                'model', 'engine', 'trim', 'exterior_color', 'interior_color'],␣
 ↪inplace=True)
```

[22]:
```
#reset index.
X.reset_index(drop=True, inplace=True)
```

[23]:
```
#creating input feature using all encoded
input_feature = pd.concat([X, encoded_df, target_encoded_col], axis=1)
input_feature.shape
```

[23]:
```
(979, 89)
```

# 1 Create model

[24]:
```
#loading library
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

[25]:
```
#spliting the train and test datafram
x_train, x_test, y_train, y_test = train_test_split(input_feature, y,␣
 ↪test_size=0.2, random_state=42)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[25]:
```
((783, 89), (196, 89), (783, 1), (196, 1))
```

[26]:
```
#fit the model and predict
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(x_train, y_train)
prediction_rfr = rfr.predict(x_test)
```

[27]:
```
#calculating R2 values.
r2_score(y_test, prediction_rfr)
```

[27]:
```
0.9188678811695565
```

[28]:
```
y.describe()
```

[28]:
```
              price
count     979.000000
mean    50202.985700
std     18700.392062
min         0.000000
25%     36600.000000
```

```
50%     47165.000000
75%     58919.500000
max    195895.000000
```

[29]:
```
#checking RMSE
print(np.sqrt(mean_squared_error(y_test, prediction_rfr)))
```

```
4977.646864101206
```

[30]:
```
#convert RMSE into error percentage
print(f'Percetage of error b/w actual and predicted price is: {np.
 ↪sqrt(mean_squared_error(y_test, prediction_rfr))/195895 * 100}')
#formulla used: RMSE/Max of traget_value * 100
```

```
Percetage of error b/w actual and predicted price is: 2.5409769846607655
```

Note: Target i.e. Price columns has min and max value are 0 and 195895. Error we are getting b/w predicted and actual price is 2.5 %. which not bad

[31]:
```
#now saving model and encoded variables
import joblib

joblib.dump(rfr, 'Random_forest_regressor.pkl')
joblib.dump(onehot_encoder, 'Onehot_encoder.pkl')
joblib.dump(target_encoder, 'Target_encoder.pkl')
```

[31]:
```
['Target_encoder.pkl']
```

### 1.1 Project Completed By Deepak Kumar

## 2 Thanks you !