Car Price Prediction

Capstone Project Report

Instructor: Dr. Rouzbeh Razavi

Authored by

Sivarangareddy Pondugula

Reg No: 811197074

Contents

Introduction:

Nowadays, machine learning is getting immensely popular in advancing technologies. It uses to develop the application's abilities and features so that many of the tasks will be automated, so neither human intervention nor an explicit program is required. This project aims to understand machine learning models' capabilities and detail the exploratory data analysis of car price prediction.

In this modern era, people tend to choose their mode of transportation. As a result public choose their automobile to commute between places. The automobile industry plays a vital role or backbone of any economy, especially the car industry, so it is essential to understand the car price mechanism.

As we know, the price of the car depends on many factors like the company's goodwill. Features of the vehicle are horsepower, mileage, and many other factors. Car price prediction is one of the elegant research areas in machine learning. In this project, we will train the car price prediction model with machine learning using python. We will be using multiple linear equations with recursive feature elimination, and we are going narrow down the feature which plays a significant role in deciding the price of the car

Problem statement:

A Chinese automobile company Geely Auto aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts.

They have contracted an automobile consulting company to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market since those may be very different from the Chinese market. The company wants to know:

- **Which variables are significant in predicting the price of a car**
- **How well do those variables describe the price of a car**

Business Goal

We are required to model the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be a good way for management to understand the pricing dynamics of a new market.

Literature:

Many researchers have done different studies or research on predicting used car prices and car prices throughout the world using different approaches and methodologies with an accuracy of 50% to 90%.

((Pudaruth, 2014) This researcher has used many other techniques in machine learning to predict the car price those are namely Multiple linear regression, Naïve Bayes, and decision trees algorithms to predict car prices based on data collected from newspapers and publications, but the author was able to get only 60 to 70 % of accuracy result, so the author has suggested using sophisticated methods to remove multicollinearity, non-significance features and an extensive set of data may have given a better result.

(Kuiper, 2008) This researcher has used the variable selection technique in his multivariate regression model so that he has selected only significant features or variables to predict the car price and reduced the complexity of the data, but his multivariate regression model is more relevant to the numerical data, and he has used only insufficient data which got it from the general motor of cars that are produced in 2005 at last authors unable to get the desired result

(Sinha, 2020) in this journal, authors have used multiple linear regression in machine learning to predict the car price. Interestingly they have removed the non-significant variables to build the model and can get the desired result of 88%, but they have not focused on outliers treatment and decided on the non-significant variables based on the correlation matrix. Finally, they missed working on multicollinearity

After considering all the above articles or research papers, car price prediction is one of the essential topics to research. By far best-achieved accuracy model is derived from the random forest Technique.

Overview of your dataset

The data set contains various factors affecting a particular car's price. There are 26 variables, namely car company, drive wheel, car-length price and many more. There is a total of 205 observations.

Importing necessary packages and functions required

In [1]:
```
#Importing necessary packages and functions required
# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
# module for performing test train split
from sklearn.model_selection import train_test_split
# Module for feture scaling
from sklearn.preprocessing import MinMaxScaler
# statmodel linear regression
import statsmodels.api as sm
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [4]:
```
car_data = pd.read_csv('CAR-PRICE.csv')
# Check the head of the dataset
car_data.head()
```

Importing data set

In [4]:
```
car_data = pd.read_csv('CAR-PRICE.csv')
# Check the head of the dataset
car_data.head()
```

Out[4]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	borerati
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.4
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.4
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.6
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.1
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.1

5 rows × 26 columns

Above table shows that there are 26 variables and 205 rows

```
In [137]: car_data.shape
Out[137]: (205, 26)
```

```
In [138]: car_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

Data Preparation and data exploration

Cleaning data

Data cleaning is an essential technique in data analytics. As we know, data comes from different sources. It may be vague and unstructured, so it is necessary to validate the data for missing values and balance data. To use the data, we should eliminate duplicates and remove the null values.

```
[140]: # percentage of missing values in each column
       round(car_data.isnull().sum()/len(car_data.index), 2)*100
```

```
Out[140]: car_ID             0.0
          symboling          0.0
          CarName            0.0
          fueltype           0.0
          aspiration         0.0
          doornumber         0.0
          carbody            0.0
          drivewheel         0.0
          enginelocation     0.0
          wheelbase          0.0
          carlength          0.0
          carwidth           0.0
          carheight          0.0
          curbweight         0.0
          enginetype         0.0
          cylindernumber     0.0
          enginesize         0.0
          fuelsystem         0.0
          boreratio          0.0
          stroke             0.0
          compressionratio   0.0
          horsepower         0.0
          peakrpm            0.0
          citympg            0.0
          highwaympg         0.0
          price              0.0
          dtype: float64
```

This shows that we don't have any null values data is good

Dropping Duplicates

Car_id is not relevant to the data, so we have dropped the Car_ID

```
# Dropping Duplicates if any
car_data=car_data.drop_duplicates()
```

```
#we can drop the car_ID variable which is not relevent for the price modeling
car_data.drop('car_ID',axis=1,inplace=True)
```

Data Understanding and Correction

The below plot exhibits the relationship between price and the remaining variables few variables have positive, and a few are negative relationships.

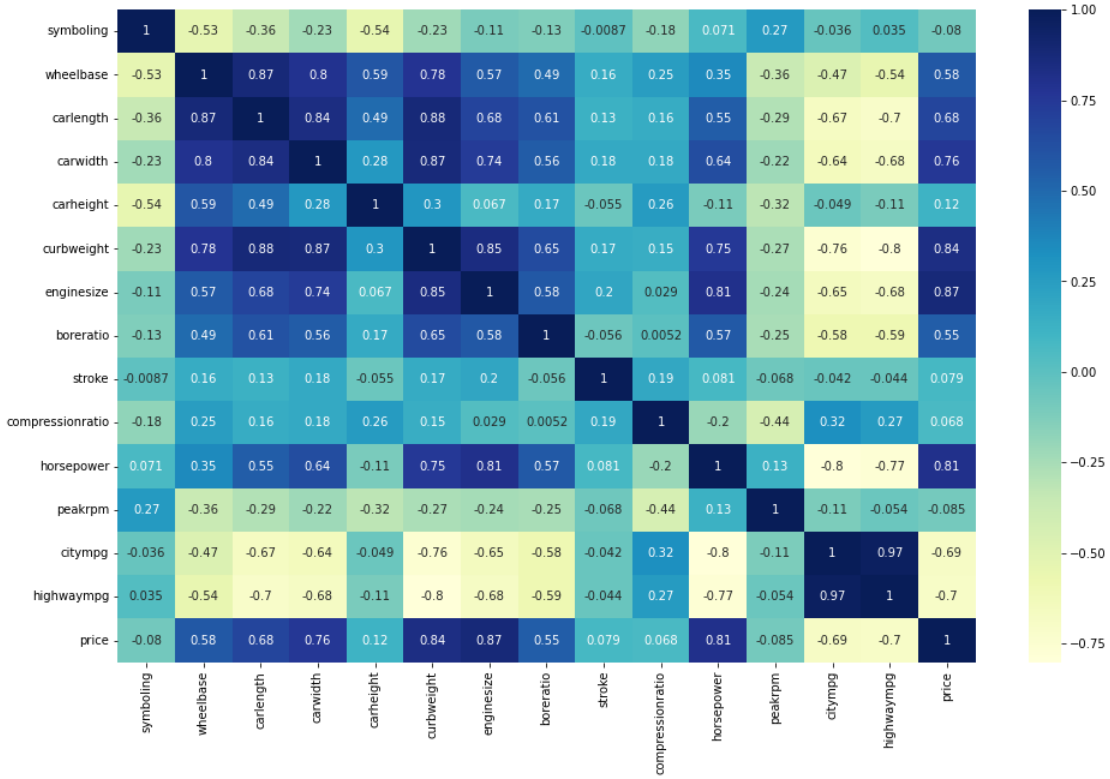
Positive relationships with the

- wheelbase,
- car length
- car width
- curb weight,
- engine size
- bore ratio,
- horsepower.

Negative relationship

- city mpg,
- highway mpg

```
In [150]:
plt.figure(figsize = (16, 10))
cor=car_data.corr()
sns.heatmap(cor, annot = True, cmap="YlGnBu")
plt.show()
```



We can also see multicollinearity between a few variables those are namely

- car length with wheelbase, car width, curb weight
- curb weight with engine size, car length, car width, wheelbase

- engine sizes with horsepower, curb weight and dimensions of the car
- highway and city mpg is highly correlated with a Pearson r coefficient of 0.97. We might choose to drop any one of these.
  We also plot some negative correlations among variables,
- curb weight horsepower with the highway/city mpg

Among all the Categorical Variables, we have modified the symbolling and car name

Symbolling, we have assigned an insurance risk rating +3 to a -3 positive sign that indicates risky, and the opposite is safe.

The car name consisted of car company and model, but we need only the company name, not the car model, so we need to consider the car company only the variable.

```
In [151]: car_data['symboling'] = car_data['symboling'].map({-2: 'safe',-1: 'safe',0: 'moderate',1: 'moderate',2: 'risky',3:'risky'})

In [152]: car_data['car_company'] = car_data['CarName'].apply(lambda x: x.split( )[0])
          car_data.drop('CarName',axis=1,inplace=True)
```

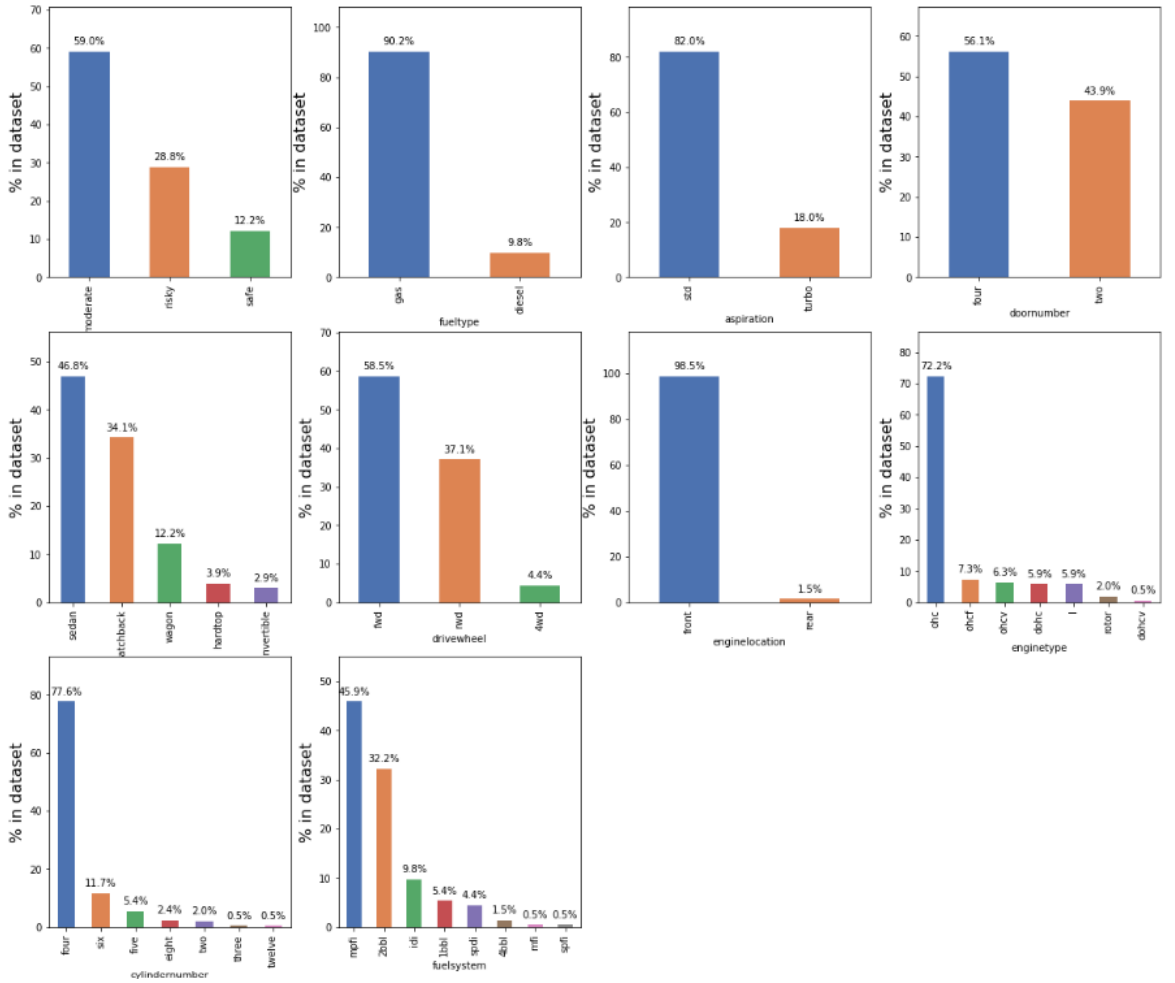
Finally, a few companies' names were misspelt. We have corrected those.

```
In [154]: # Correcting the misspelled company names.
          car_data['car_company'].replace('maxda','mazda',inplace=True)
          car_data['car_company'].replace('Nissan','nissan',inplace=True)
          car_data['car_company'].replace('porcshce','porsche',inplace=True)
          car_data['car_company'].replace('toyouta','toyota',inplace=True)
          car_data['car_company'].replace(['vokswagen','vw'],'volkswagen',inplace=True)
```

Data exploration and data preparation

We have identified a few car features that are dominant in our market

```
In [156]: # market percentage of car's with respect to features
          plot_percentages(car_data,categorical_variables[:-1])
```

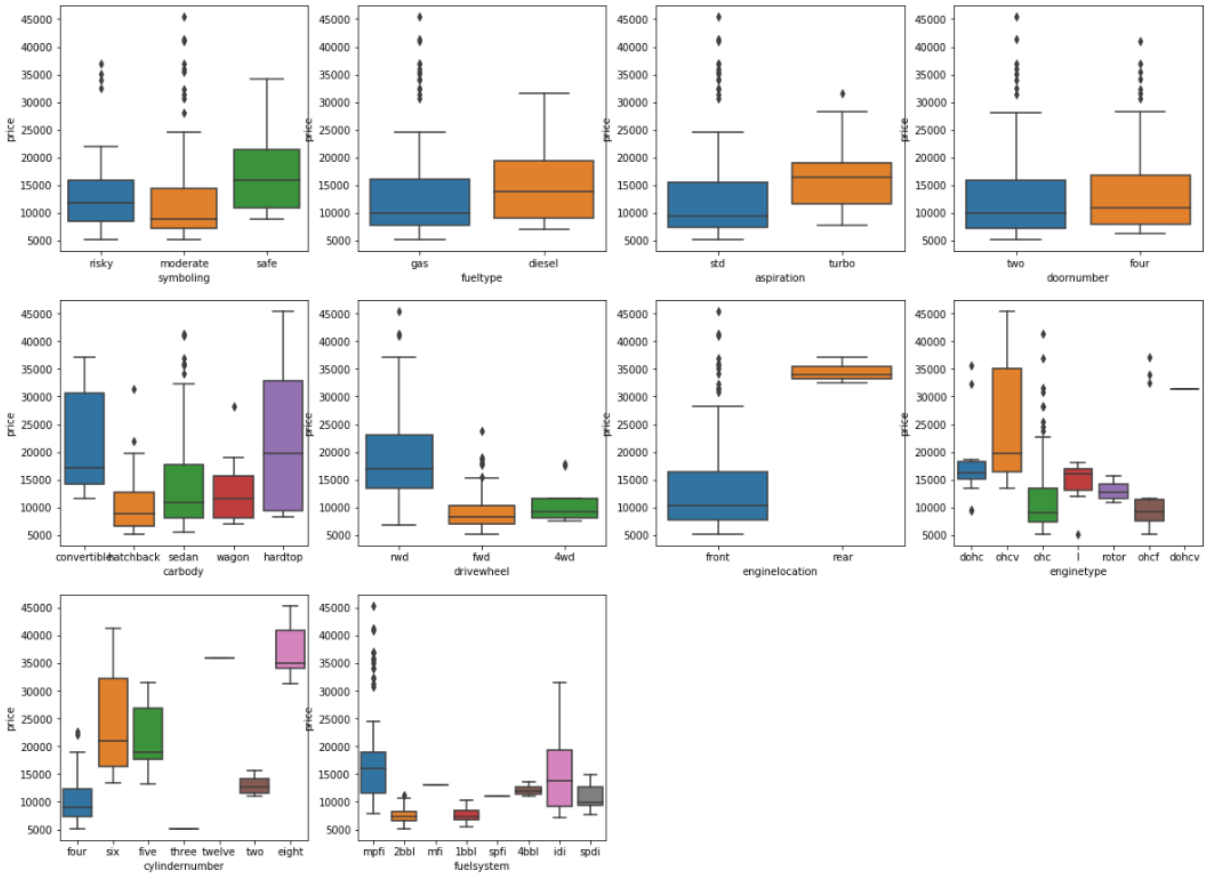


From the above plot, we can conclude that a few car features are important in our market, so Geely company should change their manufacturing line
Those features are

1. **symboling:**Moderate (0,1)
2. **Carbody:** Sedan
3. **Fuel type:** gas
4. **aspiration:** standard door numbers
5. **bers:** four
6. **drivewheel:** forward
7. **engine location:** front
8. **engine type:** ohc
9. **cylinder number:** four
10. **fuel system**: mpfi

Analysis on categorical data with price

In [26]:
```
# ploting all variables excluding car_company against the output variable
plot_cat(categorical_variables[:-1])
```



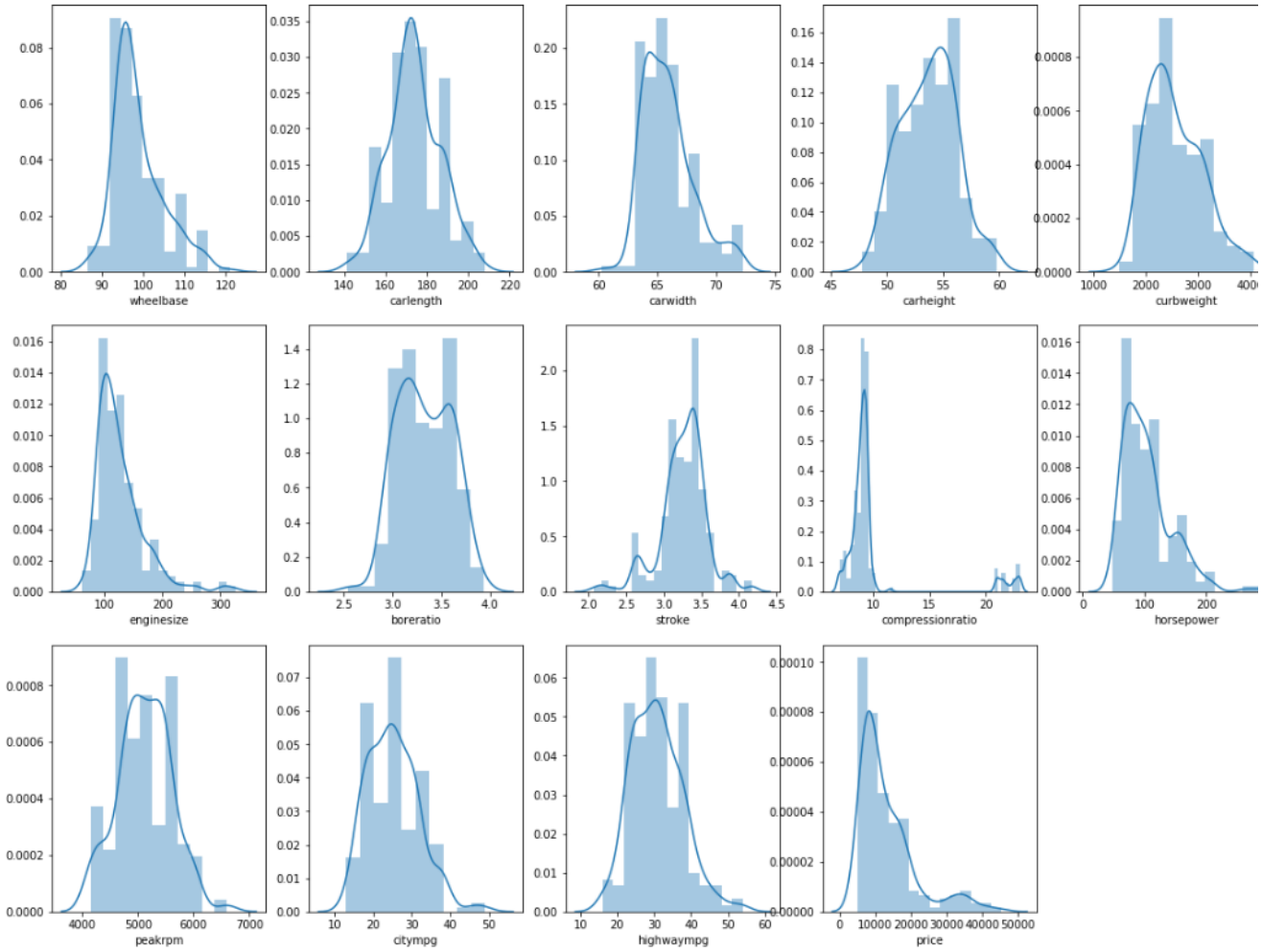
Findings

1. Fuel type, engine location and aspiration influence the pricing of the cars.
2. Cylinder number and engine type also seem to regulate the price of cars.
3. Hardtop and convertible body type cars' prices are very high compared to other regular cars

Outlier treatment

```
numeric_variables = list(car_data.columns[car_data.dtypes != 'object'])
print(numeric_variables)
plot_dist(numeric_variables)
```

```
['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'peakr
'citympg', 'highwaympg', 'price']
```



From the above distribution, we can see engine size, horsepower, and compression ratio variables to have a right-skewed distribution because of outlier

Outlier treatment

```
print(car_data[['horsepower','curbweight','enginesize']].quantile([0.01, .96]))
print(car_data[['compressionratio']].quantile([0.01, .90]))
```

```
      horsepower  curbweight  enginesize
0.01       52.12     1819.72        70.0
0.96      182.00     3657.80       209.0
      compressionratio
0.01              7.00
0.90             10.94
```

```
# Outilers in price of cars
car_data['horsepower'][np.abs(car_data['horsepower'] > 182.00)]= 182.00
car_data['horsepower'][np.abs(car_data['horsepower'] > 3657.80)]= 3657.80
car_data['enginesize'][np.abs(car_data['enginesize'] > 209.00)]= 209.00
car_data['compressionratio'][np.abs(car_data['compressionratio'] > 10.94)]= 10.94
```

We decide to treat the outliers by clipping the variables curb weight,' horsepower', and 'engine size' at 96 percentile value. And clip compression ratio at 90 percentile value, and we kept price is less than three standard deviation.

Creating new variables

Earlier in scatterplot and heatmap, we have seen few independent variables are intercorrelated, so to remove or reduce multicollinearity, we have created new variables with car length, wheelbase, car width, car weight and city/highway mpg.

```
# Creating new variable carLWratio
car_data['carLWratio'] = car_data.carlength/car_data.carwidth
# Creating new variable carWHratio
car_data['carWHratio'] = car_data.carwidth/car_data.carheight
# Creating new variable PWratio
car_data['PWratio'] = car_data.horsepower/car_data.curbweight
# Creating new variable HCmpgratio
car_data['HCmpgratio'] = car_data.highwaympg/car_data.citympg
## droping the orignal variables
car_data.drop(['carlength','carwidth','carheight','highwaympg','citympg'],axis=1,inplace=True)
```

As we know, company brand value also determines the price of the car, so we have created segments of the car

The segment has divided into three stages based on the company's mean price

- **lowtier** if company mean price is below 10,000
- **midtier** if the company mean price is above 10,000 and below 20,000

- **hightier** if the company mean price is above 20,000

In [170]:
```
company_segment_dict = {
    'cheverolet' : 'lowtier',
    'dodge' : 'lowtier',
    'plymouth' : 'lowtier',
    'honda' : 'lowtier',
    'subaru' : 'lowtier',
    'isuzu' : 'lowtier',
    'mitsubishi' : 'lowtier',
    'renault' : 'lowtier',
    'toyota' : 'lowtier',
    'volkswagen' : 'midtier',
    'nissan' : 'midtier',
    'mazda' : 'midtier',
    'saab' : 'midtier',
    'peugeot' : 'midtier',
    'alfa-romero' : 'midtier',
    'mercury' : 'midtier',
    'audi' : 'midtier',
    'volvo' : 'midtier',
    'bmw' : 'hightier',
    'buick' : 'hightier',
    'porsche' : 'hightier',
    'jaguar' : 'hightier',
    }
car_data['company_segment'] = car_data['car_company'].map(company_segment_dict)
# Dropping the orignal car_company variable
car_data.drop('car_company',axis=1,inplace=True)
car_data.head()
```

Out[170]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	curbweight	enginetype	...	stroke	compressionratio	h
0	risky	gas	std	two	convertible	rwd	front	88.6	2548	dohc	...	2.68	9.0	
1	risky	gas	std	two	convertible	rwd	front	88.6	2548	dohc	...	2.68	9.0	
2	moderate	gas	std	two	hatchback	rwd	front	94.5	2823	ohcv	...	3.47	9.0	
3	risky	gas	std	four	sedan	fwd	front	99.8	2337	ohc	...	3.40	10.0	
4	risky	gas	std	four	sedan	4wd	front	99.4	2824	ohc	...	3.40	8.0	

5 rows × 24 columns

Creating dummy variables

We need to convert the categorical variables to numeric. For this, we will use something called dummy variables. We have converted categorical variables to two levels, either 0 or 1

Handeling Categorical Variable for Linear Regression

40]:
```
# Converting categorical variables with two levels to either 1 or 0
car_data['fueltype'] = car_data['fueltype'].map({'gas': 1, 'diesel': 0})
car_data['aspiration'] = car_data['aspiration'].map({'std': 1, 'turbo': 0})
car_data['doornumber'] = car_data['doornumber'].map({'two': 1, 'four': 0})
car_data['enginelocation'] = car_data['enginelocation'].map({'front': 1, 'rear': 0})
car_data.head()
```

40]:

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	curbweight	enginetype	...	stroke	compressionratio	horsepower	peakrpm
0	risky	1	1	1	convertible	rwd	1	88.6	2548	dohc	...	2.68	9.0	111	5000
1	risky	1	1	1	convertible	rwd	1	88.6	2548	dohc	...	2.68	9.0	111	5000
2	moderate	1	1	1	hatchback	rwd	1	94.5	2823	ohcv	...	3.47	9.0	154	5000
3	risky	1	1	0	sedan	fwd	1	99.8	2337	ohc	...	3.40	10.0	102	5500
4	risky	1	1	0	sedan	4wd	1	99.4	2824	ohc	...	3.40	8.0	115	5500

```
# Creating dummy variables
df = pd.get_dummies(car_data)
# Droping 1 dummy variable and Keeping n-1 varaibles for each feature
df.drop(['symboling_risky',
         'carbody_hatchback',
         'drivewheel_4wd',
         'enginetype_l',
         'cylindernumber_three',
         'fuelsystem_1bbl',
         'company_segment_lowtier'],axis=1,inplace=True)
df.columns
```

```
]: Index(['fueltype', 'aspiration', 'doornumber', 'enginelocation', 'wheelbase',
       'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
       'horsepower', 'peakrpm', 'price', 'carLWratio', 'carWHratio', 'PWratio',
       'HCmpgratio', 'symboling_moderate', 'symboling_safe',
       'carbody_convertible', 'carbody_hardtop', 'carbody_sedan',
       'carbody_wagon', 'drivewheel_fwd', 'drivewheel_rwd', 'enginetype_dohc',
       'enginetype_dohcv', 'enginetype_ohc', 'enginetype_ohcf',
       'enginetype_ohcv', 'enginetype_rotor', 'cylindernumber_eight',
       'cylindernumber_five', 'cylindernumber_four', 'cylindernumber_six',
       'cylindernumber_twelve', 'cylindernumber_two', 'fuelsystem_2bbl',
       'fuelsystem_4bbl', 'fuelsystem_idi', 'fuelsystem_mfi',
       'fuelsystem_mpfi', 'fuelsystem_spdi', 'fuelsystem_spfi',
       'company_segment_hightier', 'company_segment_midtier'],
      dtype='object')
```

Test Train data Split

We have divided the data train size as 70 and test size as 30, and then we have applied to minimax scaler all the numeric variable scales between 0 to 1

```
from sklearn.model_selection import train_test_split

# Splitting the avilable data into training and testing set.

df_train, df_test = train_test_split(df, train_size = 0.7, test_size = 0.3, random_state = 100)
```

```
from sklearn.preprocessing import MinMaxScaler
# Using MinMaxScaler to scale all the numeric variables in the same scale between 0 and 1.
scaler = MinMaxScaler()
```

```
# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['horsepower','wheelbase','curbweight', 'enginesize', 'boreratio','carLWratio','carWHratio','PWratio','HCmpgratio

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

Modelling Approaches

Initially, we used a mixed approach. We started with linear regression with Recursive feature elimination (RFE) and variance influence factor. We ran the regression until we got all the significant factors and then used the stat model for statistical Analysis. Finally, we have also applied the model's decision tree and random forest.

This Recursive feature elimination (RFE) method eliminates the weakest or non-significant variable in the linear regression. This process will continue until it fits the model. Recursive feature elimination eliminates or reduces the dependencies and collinearity in the model.

```
y_train = df_train.pop('price')
X_train = df_train
```

```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

```
import sklearn
print(sklearn.__version__)
```

```
1.0.2
```

```
# Running RFE with the output number of the variable equal to 15
lm = LinearRegression()
lm.fit(X_train, y_train)
n_features_to_select = 15
rfe = RFE(lm, n_features_to_select=n_features_to_select)             # running RFE and selecting 15 features best describing
rfe = rfe.fit(X_train, y_train)
```

We have applied RFE feature elimination to linear regression, and then we made a summary of the linear regression, and then we found that fuelsystem_4bbl is not a significant factor so decide to drop that from train data, and we have run the model once again

Linear regression model 1

```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[col]
```

```
lm=fit_LRM(X_train_rfe)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.954
Model:                            OLS   Adj. R-squared:                  0.949
Method:                 Least Squares   F-statistic:                     173.0
Date:                Mon, 01 Aug 2022   Prob (F-statistic):           6.66e-76
Time:                        16:25:38   Log-Likelihood:                 225.25
No. Observations:                 141   AIC:                            -418.5
Df Residuals:                     125   BIC:                            -371.3
Df Model:                          15
Covariance Type:            nonrobust
================================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                          0.5136      0.069      7.457      0.000       0.377       0.650
enginelocation                -0.3728      0.049     -7.680      0.000      -0.469      -0.277
wheelbase                      0.2549      0.047      5.402      0.000       0.162       0.348
stroke                        -0.0962      0.037     -2.628      0.010      -0.169      -0.024
horsepower                     0.7717      0.078      9.929      0.000       0.618       0.926
peakrpm                        0.0829      0.031      2.706      0.008       0.022       0.144
carLWratio                    -0.0557      0.043     -1.306      0.194      -0.140       0.029
carWHratio                     0.1182      0.044      2.663      0.009       0.030       0.206
PWratio                       -0.6346      0.089     -7.162      0.000      -0.810      -0.459
carbody_convertible            0.1551      0.030      5.240      0.000       0.096       0.214
enginetype_ohc                 0.0435      0.015      2.957      0.004       0.014       0.073
cylindernumber_five           -0.0517      0.033     -1.562      0.121      -0.117       0.014
cylindernumber_four           -0.1085      0.031     -3.491      0.001      -0.170      -0.047
cylindernumber_six            -0.0511      0.030     -1.711      0.090      -0.110       0.008
fuelsystem_4bbl               -0.0468      0.042     -1.121      0.264      -0.129       0.036
company_segment_hightier       0.2152      0.022      9.628      0.000       0.171       0.259
==============================================================================
Omnibus:                        4.198   Durbin-Watson:                   2.089
Prob(Omnibus):                  0.123   Jarque-Bera (JB):                4.179
Skew:                           0.235   Prob(JB):                        0.124
Kurtosis:                       3.700   Cond. No.                         59.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In the above summary table, "fuelsystem_4bbl" has a critical value of 0.264 which is more than 0.05 so we have considered it a non-significant factor in influencing the price, so we have removed that variable and run the linear Equation once again.

Similarly, by running this loop, we have reduced many variables with less significant price determination.

In the fourth loop or fourth-time train model, we have also looked at the variance inflation factor table along with a summary we have found that

PWratio has the highest VIF and strongly correlates with horsepower, carWHratio and peak rpm. Let's remove this.

Finally, at the training for the 10th time, we found all the significant variables with low VIF

```
X_train10 = X_train9.drop('carWHratio', axis=1)
X_train10 = sm.add_constant(X_train10)
lm10 = sm.OLS(y_train,X_train10).fit()
print(lm10.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.910
Model:                            OLS   Adj. R-squared:                  0.907
Method:                 Least Squares   F-statistic:                     273.2
Date:                Mon, 01 Aug 2022   Prob (F-statistic):           9.11e-69
Time:                        16:25:40   Log-Likelihood:                 177.91
No. Observations:                 141   AIC:                            -343.8
Df Residuals:                     135   BIC:                            -326.1
Df Model:                           5
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           -0.0133      0.026     -0.505      0.614      -0.065       0.039
wheelbase                        0.3058      0.033      9.217      0.000       0.240       0.371
horsepower                       0.3750      0.032     11.579      0.000       0.311       0.439
carbody_convertible              0.1785      0.037      4.830      0.000       0.105       0.252
cylindernumber_four             -0.0686      0.019     -3.566      0.001      -0.107      -0.031
company_segment_hightier         0.3129      0.024     13.090      0.000       0.266       0.360
==============================================================================
Omnibus:                        5.403   Durbin-Watson:                   2.212
Prob(Omnibus):                  0.067   Jarque-Bera (JB):                6.579
Skew:                           0.213   Prob(JB):                       0.0373
Kurtosis:                       3.969   Cond. No.                         9.96
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
getVIF(X_train10)
```

4]:

	Features	VIF
0	const	19.89
4	cylindernumber_four	1.95
2	horsepower	1.89
5	company_segment_hightier	1.56
1	wheelbase	1.24
3	carbody_convertible	1.08

From the above picture, all the independent variables are considerably low variance inflation factor, and r squared 0.910, which means 91% and adjusted r squared is 90.7% relatively high which has significantly fit to the model

Now we are good with train data. It is time to look at test data before making conclusions about equations.

Scaling to test data

```
num_vars=num_vars = ['horsepower','wheelbase','curbweight', 'enginesize', 'boreratio','carLWratio','carWHratio','PWratio','HC
df_test[num_vars] = scaler.transform(df_test[num_vars])
```

```
y_test = df_test.pop('price')
X_test = df_test
```

```
# Now let's use our model to make predictions.

# Creating X_test_new dataframe by dropping variables from X_test
X_train10= X_train10.drop(['const'], axis=1)
X_test_new = X_test[X_train10.columns]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)
```

```
# Making predictions
y_pred = lm10.predict(X_test_new)
```

We have applied the test data scaling to the variable only used for the train set.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
print('Model RMSE:',rmse)

from sklearn.metrics import r2_score
r2=r2_score(y_test, y_pred)
print('Model r2_score:',r2)
```

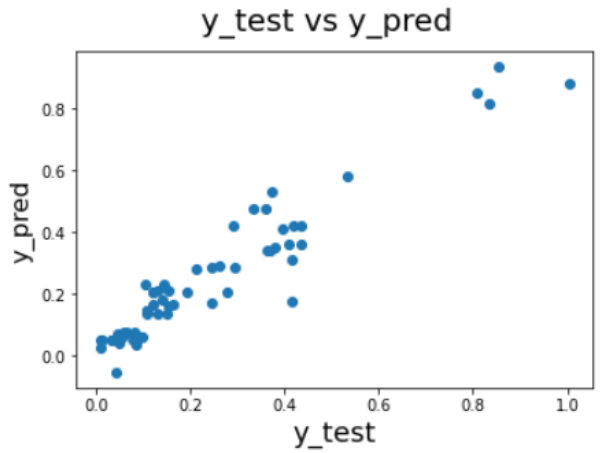
```
Model RMSE: 0.0673924212933167
Model r2_score: 0.9056388935908385
```

From this, we can conclude that the r squared value on test data is almost equal to the trained adjusted r squared and has a low RMSE of 0.067.

Performance evaluation

```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)
```

```
Text(0, 0.5, 'y_pred')
```



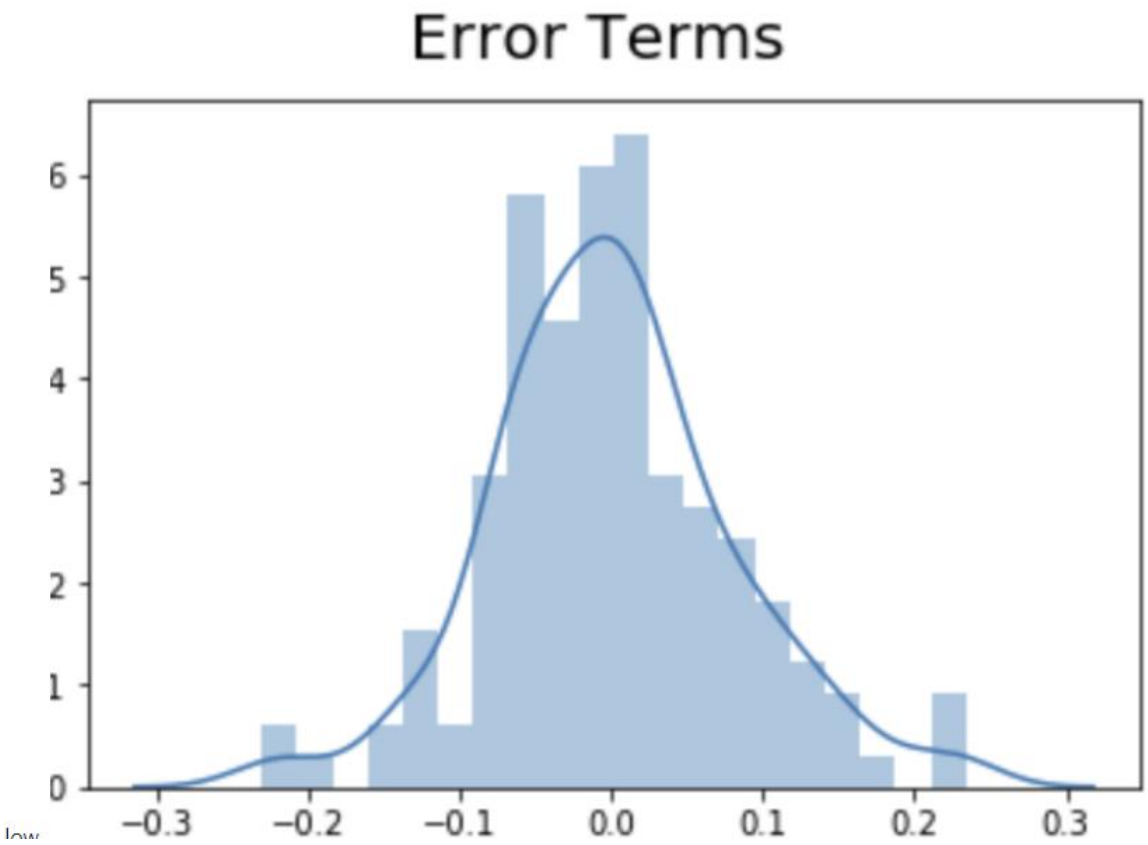
As we can see, the y test and y prediction are almost linear with some variations overall. It is a pretty good linear spread.

Residual Analysis of the train data

```
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
```

```
Text(0.5,0,'Errors')
```



As we know, it looks like not perfect normal distribution although centred is almost perfectly zero. However, still, some variance is not explained. Considering the size of the test data and train, this error term is almost normally distributed.

Finally, we are evaluating the model with tests, and train data R squared value and adjusted R-squared value

```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
print('Model RMSE:',rmse)

from sklearn.metrics import r2_score
r2=r2_score(y_test, y_pred)
print('Model r2_score:',r2)
```

```
Model RMSE: 0.06739242129331674
Model r2_score: 0.9056388935908384
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.910
Model:	OLS	Adj. R-squared:	0.907
Method:	Least Squares	F-statistic:	273.2
Date:	Sun, 25 Nov 2018	Prob (F-statistic):	9.11e-69
Time:	20:19:17	Log-Likelihood:	177.91
No. Observations:	141	AIC:	-343.8
Df Residuals:	135	BIC:	-326.1
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0133	0.026	-0.505	0.614	-0.065	0.039
wheelbase	0.3058	0.033	9.217	0.000	0.240	0.371
horsepower	0.3750	0.032	11.579	0.000	0.311	0.439
carbody_convertible	0.1785	0.037	4.830	0.000	0.105	0.252
cylindernumber_four	-0.0686	0.019	-3.566	0.001	-0.107	-0.031
company_segment_hightier	0.3129	0.024	13.090	0.000	0.266	0.360

Omnibus:	5.403	Durbin-Watson:	2.212
Prob(Omnibus):	0.067	Jarque-Bera (JB):	6.579
Skew:	0.213	Prob(JB):	0.0373
Kurtosis:	3.969	Cond. No.	9.96

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

As we know, the higher the R-squared value better fit the model. Since both test and train have higher R squared values, there is less variation or difference between observed and predicted values.

Random forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
```

```
Rf = RandomForestRegressor(n_estimators = 15,
                           criterion = 'mse',
                           random_state = 20,
                           n_jobs = -1)
Rf.fit(X_train,y_train)
Rf_train_pred = Rf.predict(X_train)
Rf_test_pred = Rf.predict(X_test)


r2_score(y_test,Rf_test_pred)
```

```
0.9295879529474357
```

As we can see we have we have also applied random forest to the same data we got an accuracy as 0.92 or 92.9%

Decision tree

```
from sklearn.tree import DecisionTreeRegressor
```

```
dt_regressor = DecisionTreeRegressor(random_state=0)
dt_regressor.fit(X_train,y_train)
y_train_pred = dt_regressor.predict(X_train)
y_test_pred = dt_regressor.predict(X_test)
dt_regressor.score(X_test,y_test)
```

```
]: 0.8732620068024589
```

By applying to a decision tree to the same data we got only 0.87% accurate or 87%

Conclusion

Final model evaluation
In the end, we can get a variable with low VIF and p values. This variable explains the price to an great extent.

Predictor	Coef	p-value
wheelbase	0.3058	0.000
horsepower	0.3750	0.000
carbody_convertible	0.1785	0.000
cylindernumber_four	-0.0686	0.001
company_segment_hightier	0.3129	0.000

The best-fitted Equation is going to be

Price = -0.0133 + 0.3058 *wheelbase +0.3750*horsepower+0.1785* Car body Convertible -0.0686 * cylinder number four +0.3129* company segment high tier

This Equation indicates how the price or output changes when there is a change in the independent var iable
Suppose if there is one unit change in wheelbase, 0.3058 will increase by price similarly to all other va riables except negative coefficient cylinder number four. In the United States, cylinder_number_four is a mostly available feature, so having it relatively lowers the price of the car.

As we can see we have applied all three regression models to data we found random forest is best with an accuracy of 92.9%.