Used Car Price Prediction

Objective

The main objective of this analylsis is to apply Supervised Machine Learning techniques such as One-Hot encoding, polynomial feature engineering, data scaling and Ridge Regression regularization to train models and observe how data preparation can affect a given score of a model.

Data Source: https://www.kaggle.com/billumillu/predicting-costs-of-used-cars-machinehack (https://www.kaggle.com/billumillu/predicting-costs-of-used-cars-machinehack)

In [49]:
```
import pandas as pd
data =  pd.read_csv("train-data.csv")
carData = data.copy()
```

Data Summary

The data is comprised of a list of used cars where 'Price' will be used as the dependent variable. There are 6019 records in the dataset and 14 columns.

In [2]:
```
data.head(3)
```

Out[2]:

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First

Price - Dependent Variable: Numerical value where vehicle is priced
Name - Categorical Variable: Name of the car
Location - Categorical Variable: City where car is held
Year - Year of the car model
Kilometers_Driven - Numerical Values: Number of kilometers the vehicle has been driven
Fuel_Type - Categorical Variable: Engine type of the vehicle
Transmission - Categorial Variable: Transmission type
Owner_type - Numerical Value: Number of owners
Mileage - Numerical Values: KMPL Kilometer Per Liter
Engine - Numerical Values: Engine is described by CC (cubic centimeters)
Power - Numerical Values: Described as BHP (brake horse power)
Seats - Numerical Values: Number of seats the vehicle has
New_Price - Numerical Value: If available, the price of the vehicle if it were new

Data Cleaning and Feature Engineering Summary

- Removed unnecessary columns. Ex. The "Unnamed: 0" column is just an index.
- Converted all numerical values to floats and removed strings.
- Filled in missing values for "Mileage", "Engine", "Power", "Seats" and "New_Price"
- Tested the target variable "Price" for normality and tried log and boxcox transformations without any luck getting it to pass a normal test, but still used the transformed variable when trainig models.
- Applied One-Hot encoding tranformations along with polynomial feature engineering.

Summary of Linear Regression Models

I started with a simple Linear Regression Model that took in all features as either untouched floats or one-hot encoded columns for the unordered categorical variables. The data was split into two sets where 75% of the data was dedicated for training and the remaining 25% of the data for testing. I used two scoreing metrics to guage the power of the models with hopes to better understand the data and how transformations affect the score.

The second model included polynomial feature engineering. After the categorical variables were one-hot encoded, the remaining quantitative features were used calculate

1st Model Scores
Mean Squared Error: 28.28450804227052
R2 Score: 0.7553591337349053

2nd Model Scores
Mean Squared Error: 16.436009701540872
R2 Score: 0.8578402125532008

3rd Model Scores
Mean Squared Error: 16.27299108354964
R2 Score: 0.8592502075887556

Based on the scores above, the most useful model for predicting used car price would be the final model that used Ridge regression regularization techniques as well as polynomial feature engineering.

Next steps

Some of the new price imputations could be improved upon. Higher-end cars like Bently and Lamborghini should not have new prices in the same range as the average car. Insead of using the total average to impute new_prices, I could rather do independent research and manually fill in the values for the vehicles with missing information.

In the future, I could introduce a pipeline of instruction to streamline model creating and preprocessing steps for quicker data preparation when introducting new models.

Inital Data Cleaning

In [3]:
```
#identify missing values
data.isnull().sum()
```

Out[3]:
```
Unnamed: 0              0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 36
Power                  36
Seats                  42
New_Price            5195
Price                   0
dtype: int64
```

In [4]:
```
#drop "Unnamed Column"
carData.drop("Unnamed: 0", axis=1, inplace=True)

#get first word in car name. This will drop the number of unique values from 1
876 to 31
carData.Name = carData.Name.apply(lambda x: x.split()[0])
```

Convert Strings to Floats

In [52]:
```
import re
#remove strings from all numerical values and convert to float.
carData.Mileage = carData.Mileage.str.extract('(\d*\.\d+|\d+)').astype(float)
carData.Engine = carData.Engine.str.extract('(\d*\.\d+|\d+)').astype(float)
carData.Power = carData.Power.str.extract('(\d*\.\d+|\d+)').astype(float)
carData.New_Price = carData.New_Price.fillna(0)
carData.New_Price = carData.New_Price.str.extract('(\d*\.\d+|\d+)').astype(flo
at)
```

Impute missing New_Price

In [53]:
```
import math
#create a dictionary that will store avg New_Price values for specific car Nam
es.
name_dict = {}

#lowercase all names to eliminate duplicate names like isuzu and ISUZU
carData.Name = carData.Name.str.lower()

#for each unique car name, find the average and store in the name_dict diction
ary
for name in carData.Name.unique():
    if math.isnan(carData.New_Price[carData.Name == name].mean()):
        name_dict[name] = carData.New_Price.mean()
    else:
        name_dict[name] = carData.New_Price[carData.Name == name].mean()
```

In [54]:
```
#dictionary of cars and their avg new_price. If there wasnt a new_price avg av
ailable, the avg new_price of the data (20.32) was used instead.
#name_dict
```

In [56]:
```
#iterate through new_price column and fill NaNs with avg value for the car nam
e.
for index,row in carData.iterrows():
    name = row['Name']
    if math.isnan(row['New_Price']):
        carData['New_Price'][index] = name_dict[name]
```

Fill in all blanks and 0s with mean or median of data

In [9]:
```
meanMileage = carData.Mileage.mean(skipna=True)
carData.Mileage = carData.Mileage.fillna(0)
carData.Mileage = carData.Mileage.replace(0,meanMileage)

medianEngine = carData.Engine.median(skipna=True)
carData.Engine = carData.Engine.fillna(medianEngine)

medianPower = carData.Power.median(skipna=True)
carData.Power = carData.Power.fillna(medianPower)

meanSeats = round(carData.Seats.mean(skipna=True))
carData.Seats = carData.Seats.fillna(meanSeats)
carData.Seats = carData.Seats.replace(0,meanSeats)
```

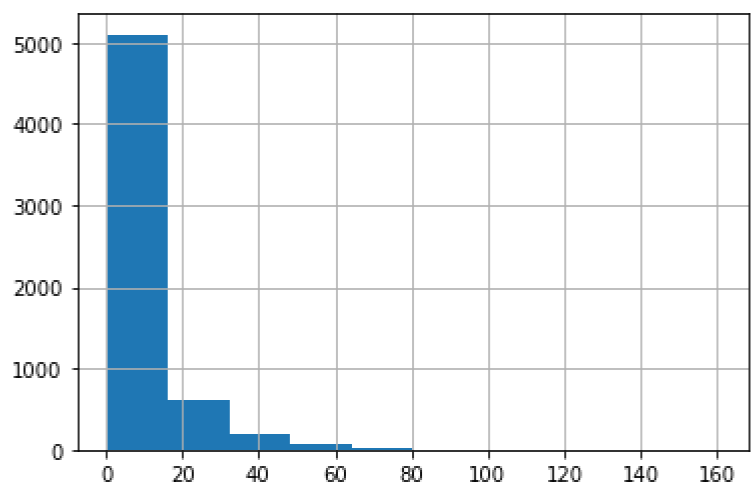
In [10]:
```
#check df again for null values
carData.isnull().sum()
```

Out[10]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
New_Price            0
Price                0
dtype: int64
```

Transform the Target Variable

In [11]:
```
#the dependent variable is skewed right.
carData.Price.hist()
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x168c35d2ca0>
```

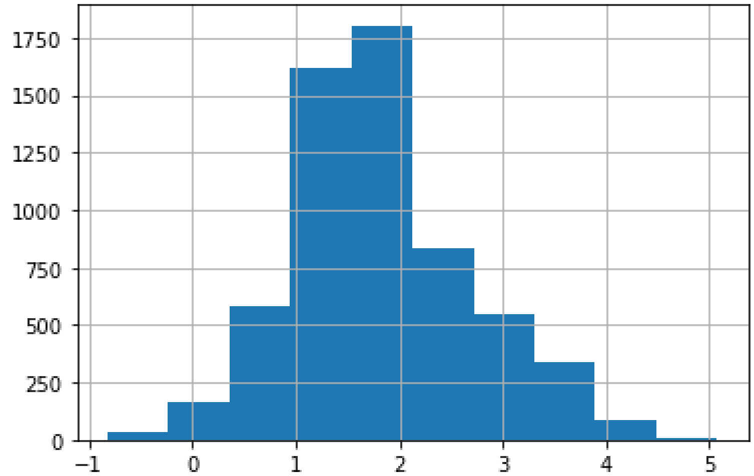


In [12]:
```
from scipy.stats.mstats import normaltest
#Frequentist statisticians would say that you accept that the distribution is normal
#(more specifically: fail to reject the null hypothesis that it is normal) if p > 0.05.
normaltest(carData.Price.values)
```

Out[12]:
```
NormaltestResult(statistic=4386.365238467286, pvalue=0.0)
```

In [13]:
```
import numpy as np
logPrice = np.log(carData.Price)
logPrice.hist()
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x168c5a32790>
```



In [14]:
```
#fails the normal test but looks much more normal. Possible error due to large sample size.
normaltest(logPrice.values)
```

Out[14]:
```
NormaltestResult(statistic=168.27590227342037, pvalue=2.879732201025881e-37)
```

One-hot encode all categorical variables

In [15]:
```
#some features like Seats and Year might be usefull to ohc? For now its just
mask = carData.dtypes == np.object
categorical_cols = carData.columns[mask]

from sklearn.preprocessing import OneHotEncoder, LabelEncoder
#copy the data. will append encoded data back to og df later.
data_ohc = carData.copy()

le = LabelEncoder()
ohc = OneHotEncoder()

for col in categorical_cols:
    #integer encode the string categories i.e. Mon, Tues will be encoded to 1,
2 etc
    dat = le.fit_transform(data_ohc[col]).astype(np.int)
    #drop the original col
    data_ohc = data_ohc.drop(col,axis=1)
    #one hot encode the data! --this will return a sparse array to save memory
when encoding larger datasets
    new_dat = ohc.fit_transform(dat.reshape(-1,1))
    #create unique column names
    n_cols = new_dat.shape[1]
    #list comprehension ftw
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]
    #Create new df and use toArray() to turn our sparse matrix into a more rea
dable format
    new_df = pd.DataFrame(new_dat.toarray(), index=data_ohc.index, columns=col
_names)
    #append the new data to the dataframe
    data_ohc = pd.concat([data_ohc, new_df],axis=1)
```

In [16]:
```
#data_ohc
```

Develop Models

In [17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import (StandardScaler, PolynomialFeatures)
```

Create X and y

In [18]:
```
y = 'Price'
X = data_ohc.drop(y,axis=1)
y = data_ohc[y]
```

1st model

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rand
om_state=999)

lr = LinearRegression()
LR = lr.fit(X_train,y_train)

y_test_pred = LR.predict(X_test)
```

Baseline Scores

In [20]:
```
print('Mean Squared Error: ',mean_squared_error(y_test, y_test_pred))
print('R2 Score: ',r2_score(y_test, y_test_pred))
```

```
Mean Squared Error:  28.28450804227052
R2 Score:  0.7553591337349053
```

2nd Model - Add polynomial features, rerun LR model and compare to baseline scores

In [33]:
```
data_poly = data_ohc.copy()
poly_features = data_poly[['Year', 'Kilometers_Driven', 'Mileage','Engine', 'P
ower','Seats']]

pf = PolynomialFeatures(degree=2,include_bias=False)
temp_poly_data = pf.fit_transform(poly_features)
temp_poly_data = pd.DataFrame(temp_poly_data)
#data_poly = data_poly.drop(poly_features, axis=1)
data_poly = pd.concat([data_poly, temp_poly_data], axis=1)
data_poly

y = 'Price'
X = data_poly.drop(y,axis=1)
y = data_poly[y]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rand
om_state=999)
lr = LinearRegression()
LR = lr.fit(X_train,y_train)
y_test_pred = LR.predict(X_test)

print('Mean Squared Error: ',mean_squared_error(y_test, y_test_pred))
print('R2 Score: ',r2_score(y_test, y_test_pred))
```

```
Mean Squared Error:  16.436009701540872
R2 Score:  0.8578402125532008
```

3rd Model - apply Standard Scaler and run Ridge regression instead

In [43]:
```
# The ridge regression model
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

y = 'Price'
X = data_poly.drop(y,axis=1)
y = data_poly[y]

sc = StandardScaler()
data_scaled_poly = X.copy()
data_scaled_poly = sc.fit_transform(data_scaled_poly)

X_train, X_test, y_train, y_test = train_test_split(data_scaled_poly, y, test_
size=0.25, random_state=999)
```

In [48]:
```
rr = Ridge(alpha=0.001)
rr = rr.fit(X_train, y_train)
y_pred_rr = rr.predict(X_test)

print('Mean Squared Error: ',mean_squared_error(y_test, y_pred_rr))
print('R2 Score: ',r2_score(y_test, y_pred_rr))
```

```
Mean Squared Error:  16.27299108354964
R2 Score:  0.8592502075887556
```