Supervised Machine Learning: Regression
IBM Skills Network

Project Report

By:
Than Phuc Hau

TABLE OF CONTENTS

TABLE OF CONTENTS **2**
Main Objective **3**
Brief Description **3**
Initial plan **4**
Data cleaning & Feature engineering **4**
Check for any duplicates 4
Check for missing value and datatype 5
Check for correlated features 6
Calculate skewness of data 6
Process unused columns 7
Encode categorical columns 8
Define features and target 9
Linear Regression Model **9**
Recommended Model **11**
Key Findings and Insights **12**
Suggestions **13**
Summary **13**

Main Objective

As the quality of life of humans has improved a lot in recent years, the demand for mid-high priced items such as cars has also increased. With the remarkable development of technology and automation, there are a lot of car branches along with different technology, production, operation and quality. However, not everyone has enough knowledge on this field, leading to difficulty in choosing a car with reasonable price.

This project aims to estimate (prediction) the price of a car according to some of its details.

Brief Description

This dataset has 26 features

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...
...	...	...	...	...	...	...	...	...	...	...	...
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...

205 rows × 26 columns

The features are **car_ID**, **symboling**, **CarName**, **fueltype**, **aspiration**, **doornumber**,... with 10 features are categorical and 16 features are numerical.

Some features we will not use such as car_ID and CarName, we will drop or extract useful details in these columns. We will also encode categorical features to use for the model.

In the next sections, we shall explore more information about the underlying patterns in this data set.

Initial plan

The plan would go as follows:

- Check for duplicates and deal with any
- Check for missing values and deal with any
- Calculate correlation values and draw heatmap for visualization
- Check for skewness of data
- Encode categorical data
- Use different models for training and select best model
- Access and visualize the result

Data cleaning & Feature engineering

Check for any duplicates

```
df.car_ID.is_unique
```

```
True
```

There is no duplicated value.

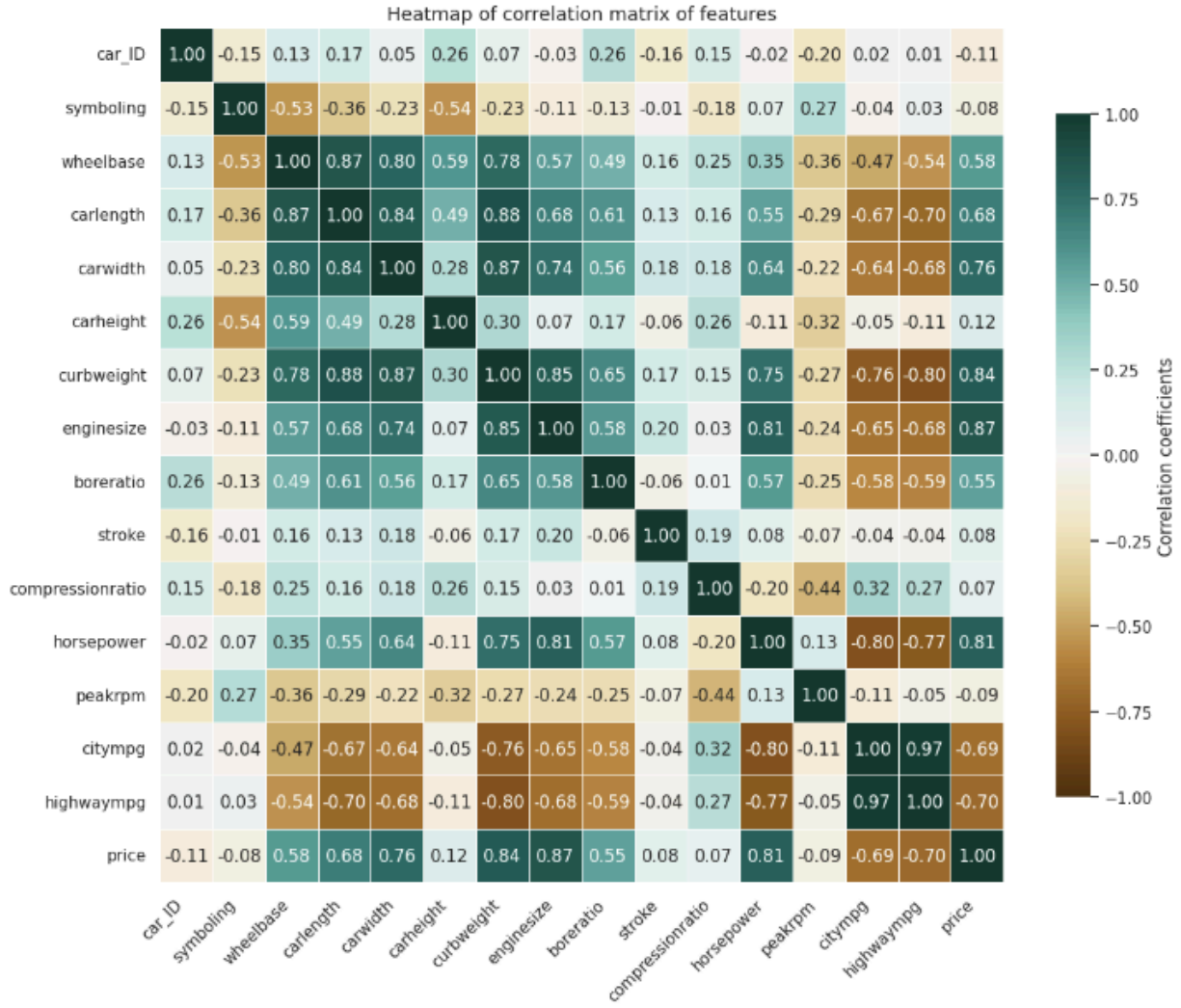
Check for missing value and datatype

```
df.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
```

There is no missing value in these columns, so we will bypass this step.

Check for correlated features



When accessing correlated features, we can start by calculating correlation matrix and visualize it by a heat map.

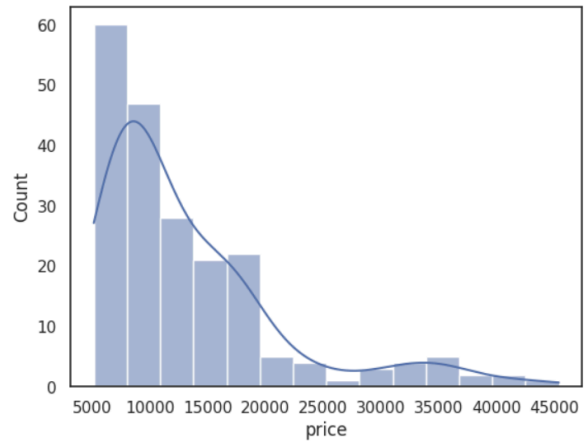
According to the heat map, some features show strong relations. For example, the curbweight (weight of the car) and highwaympg (highway fuel consumption) show a strong negative correlation (-0.8), which indicates the heavier the car is, the more fuel it will consume.

Calculate skewness of data

For better performance, we need to ensure that our target Price is not heavily skewed. We will check the skewness of the target column ‘price’ and correct the skewness if necessary.

```
plot = sns.histplot(data = df, x=df['price'], kde=True);
print(f"Skewness = {df['price'].skew()}")
```

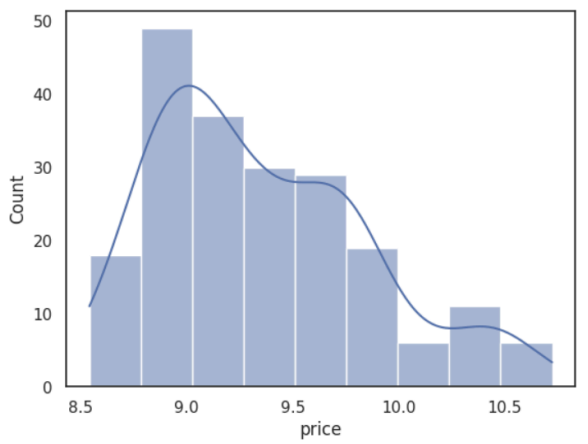
```
Skewness = 1.7776781560914454
```



Before

```
log_transformed = np.log(df['price'])
plot = sns.histplot(data = df, x=log_transformed, kde=True);
print(f"Skewness = {log_transformed.skew()}")
```

```
Skewness = 0.672888533977329
```



After

We can see that our target before was heavily skewed with skewness up to -1.77, after processing by log transform, the skewness is -0.67, which is acceptable.

Process unused columns

As mentioned before, we will drop column car_ID when creating features variable X for our model. Besides, we will extract Brand name from column CarName, the other details are unusable.

```
df.drop(columns='car_ID', axis=0, inplace = True)
```

```
# Create a new column 'brand' by extracting the brand name from 'CarName'
df['brand'] = df.CarName.str.split(' ').str.get(0).str.lower()
```

```
# Check the unique brand names
df.brand.unique()
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

```
# Correct the misspelled brand names
df['brand'] = df['brand'].replace(['vw', 'vokswagen'], 'volkswagen')
df['brand'] = df['brand'].replace(['maxda'], 'mazda')
df['brand'] = df['brand'].replace(['porcshce'], 'porsche')
df['brand'] = df['brand'].replace(['toyouta'], 'toyota')
```

```
# Check the unique brand names
df.brand.unique()
```

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

```
df.drop(columns='CarName', axis=0, inplace = True)
```

Encode categorical columns

We will one-hot encoding for categorical columns so that we can use them for our models. There will be 10 columns that need to be encoded. Also, we also drop first columns to avoid dummy variable traps.

```
Columns that will be applied One-Hot Encode: ['fueltype', 'aspiration', 'doornumber', '

 DataFrame apter applying One-Hot Encoding:
   symboling  wheelbase  carlength  carwidth  carheight  curbweight  \
0          3       88.6      168.8      64.1       48.8        2548
1          3       88.6      168.8      64.1       48.8        2548
2          1       94.5      171.2      65.5       52.4        2823
3          2       99.8      176.6      66.2       54.3        2337
4          2       99.4      176.6      66.4       54.3        2824

   enginesize  boreratio  stroke  compressionratio  ...  brand_nissan  \
0         130       3.47    2.68               9.0  ...           0.0
1         130       3.47    2.68               9.0  ...           0.0
2         152       2.68    3.47               9.0  ...           0.0
3         109       3.19    3.40              10.0  ...           0.0
4         136       3.19    3.40               8.0  ...           0.0

   brand_peugeot  brand_plymouth  brand_porsche  brand_renault  brand_saab  \
0            0.0             0.0            0.0            0.0         0.0
1            0.0             0.0            0.0            0.0         0.0
2            0.0             0.0            0.0            0.0         0.0
3            0.0             0.0            0.0            0.0         0.0
4            0.0             0.0            0.0            0.0         0.0

   brand_subaru  brand_toyota  brand_volkswagen  brand_volvo
0           0.0           0.0               0.0          0.0
1           0.0           0.0               0.0          0.0
2           0.0           0.0               0.0          0.0
3           0.0           0.0               0.0          0.0
4           0.0           0.0               0.0          0.0

[5 rows x 65 columns]
```

Define features and target

After that, we will divide our data set into our features ‘X’ and target ‘y’. Calculating polynomial features will be done in the next section since we will be using a pipeline.

```
data = df_final.copy()
y_col = data['price']
X = data.drop(columns='price',axis=0)
Y = data['price']
print('X shape: ',X.shape)
print('Y shape: ',Y.shape)
```

```
X shape:  (205, 64)
Y shape:  (205,)
```

Linear Regression Model

We will use 3 regression models and determine which is best for the data set provided. Our approach would be constructing a cross-validation grid-search that would find the best hyperparameters for each of the models then we will compute the R^2 score for each of them as well as the RMSE and compare between each of the models at the end.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

estimator_lr = Pipeline([("polynomial_features",PolynomialFeatures()),
                         ("scaler",StandardScaler()),
                         ("linear_regression",LinearRegression())])
estimator_las = Pipeline([("polynomial_features",PolynomialFeatures()),
                         ("scaler",StandardScaler()),
                         ("las_regression",Lasso())])
estimator_r = Pipeline([("polynomial_features",PolynomialFeatures()),
                         ("scaler",StandardScaler()),
                         ("ridge_regression",Ridge())])
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
```

```
kf = KFold(shuffle=True, random_state=42, n_splits=3)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2, random_state=40)
```

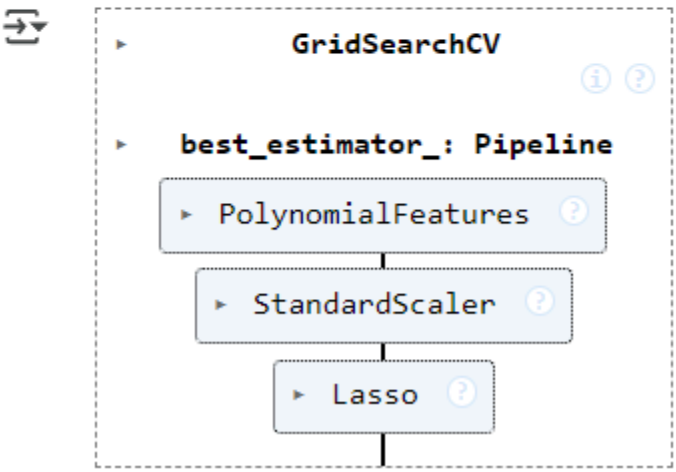
```
params_r = {'polynomial_features__degree': [1,2,3],
            'ridge_regression__alpha': np.geomspace(0.1,50,20)}
params_las = {'polynomial_features__degree': [1,2,3],
            'las_regression__alpha': np.geomspace(0.1,50,20)}
params_lr = {'polynomial_features__degree': [1,2,3]}
```

The models we will be using are **LinearRegression(), Lasso(), and Ridge()**. We constructed our pipeline estimators for each of them and loaded with them a standard scaler and polynomial features.

The parameters we intend to use for the grid-search will change the degree of the polynomial features and also change the alpha values for both the Lasso and Ridge models only.

We also defined our cross-validation object and instantiated a K-fold cross-validation object with shuffle=True and K=3. This would try to decrease over-fitting our models to the data set.

```
from sklearn.model_selection import GridSearchCV
grid_r = GridSearchCV(estimator_r, params_r, cv=kf)
grid_r.fit(X_train, Y_train)

grid_lr = GridSearchCV(estimator_lr, params_lr, cv=kf)
grid_lr.fit(X_train, Y_train)

grid_las = GridSearchCV(estimator_las, params_las, cv=kf)
grid_las.fit(X_train, Y_train)
```



Recommended Model

To determine which of our models is best, we will try to calculate a few metrics about each model to see if they managed to successfully beat the other models.

We will be using `r2_score()` and `mean_square_error()` from the `sklearn.metrics` library. We first construct a function that would help us compute the RMSE instead of the MSE shown below.

```
def rmse(Y_pred, Y_test):
    return np.sqrt(mean_squared_error(Y_test, Y_pred))
```

After that, we will use 3 trained models to predict on our test set and calculate R2 score and RMSE of 3 models to assess.

Then, we would decide which of the models is the best based on those results.

```
# Predict
Y_pred_lr = grid_lr.predict(X_test)
Y_pred_las = grid_las.predict(X_test)
Y_pred_r = grid_r.predict(X_test)

# Calculate R2 score and RMSE
r2_lr = r2_score(Y_test, Y_pred_lr)
r2_las = r2_score(Y_test, Y_pred_las)
r2_r = r2_score(Y_test, Y_pred_r)

rmse_lr = rmse(Y_pred_lr, Y_test)
rmse_las = rmse(Y_pred_las, Y_test)
rmse_r = rmse(Y_pred_r, Y_test)
```

```
-=Linear Regression=-
R2 score: 0.775202886035118
RMSE: 2295.3675774937824

-=Lasso Regression=-
R2 score: 0.8714623818966186
RMSE: 1735.6893597862468

-=Ridge Regression=-
R2 score: 0.9183450025538463
RMSE: 1383.4015629157439
```

As we can see, Linear regression is not as good as other models. In contrast, both linear and ridge regressions have relatively high R^2 scores. Note that our target 'price' has value in the range 5000~45000, so the RMSE 1735 and 1383 are reasonable.

Based on these results the best choice would be Ridge Regression.

Key Findings and Insights

In this section we would dive deeper into the construction of each of the models. We would look into the values and insights about the weights of each of the three models used.

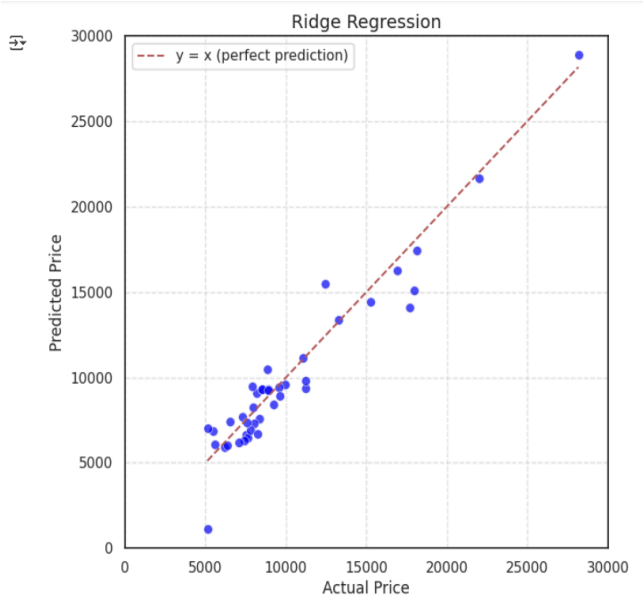
First let's look into the weights given by each model for each feature.

```
Best degree (Linear): 1
Best degree (Lasso): 2
Best degree (Ridge): 2

=== Linear Regression Coefficients ===
     Feature   Coefficient
0          1 -1.310827e-10
1  symboling -2.544689e+02
2  wheelbase  1.589765e+03
3  carlength -1.801497e+03
4   carwidth  1.832355e+03
Number of coefficients: 65
```

```
=== Lasso Regression Coefficients ===
     Feature  Coefficient
0          1          0.0
1  symboling          0.0
2  wheelbase          0.0
3  carlength          0.0
4   carwidth          0.0
Number of coefficients: 2145

=== Ridge Regression Coefficients ===
     Feature  Coefficient
0          1     0.000000
1  symboling    21.645413
2  wheelbase   105.415920
3  carlength    98.260977
4   carwidth   188.992042
Number of coefficients: 2145
```

As we can see, Lasso has 5 zero weights which is expected from a lasso regression model. In contrast, ridge regression has only 1 zero weight.

Surprisingly, as we can see our simple linear regression model did not over-fit and does not have large weight values this is most probably due to using cross-validation. The next figure would show our Ridge regression model estimates of the test data.



Suggestions

The car price dataset gives us many features to test different models. We can try ElasticNet with cross-validation to get a better balance between bias and variance. We should also review our models to understand which features affect the price the most. In addition, we can use GridSearchCV with wider parameter ranges to find better model settings

Summary

Predicting car prices helps companies focus on features that increase value and helps buyers know fair prices. This dataset has good potential for more analysis. Even though more tuning can be done, our project followed the plan and gave useful insights about how car features relate to price.