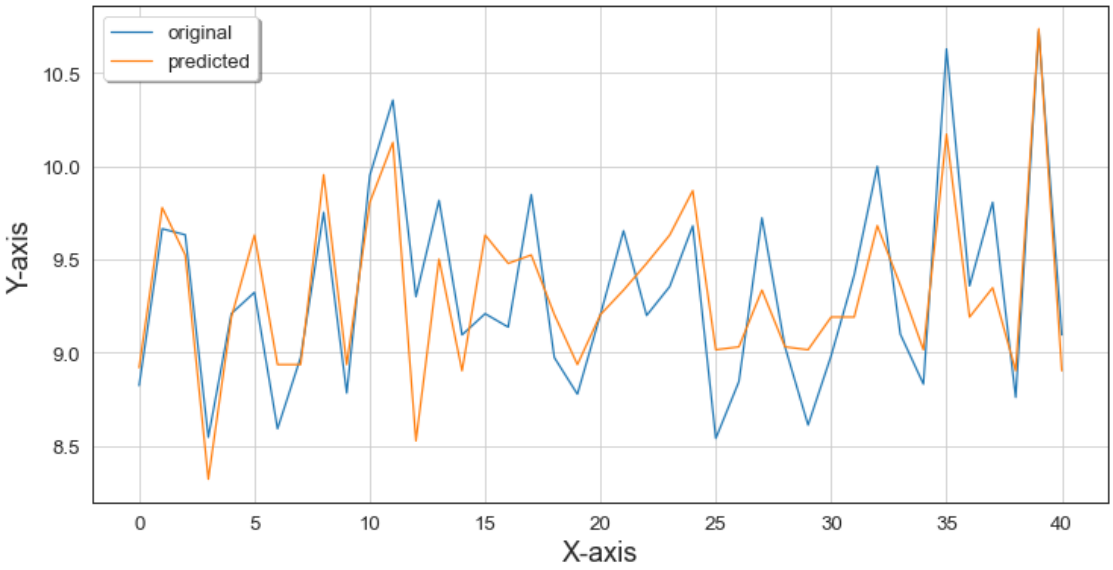



# Car Price Prediction

AI PROJECT REPORT

Faeze Shakouri | 9632441 | Artificial Intelligence | July 2021

## 3. Is this supervised or unsupervised learning? Why? **PDF**

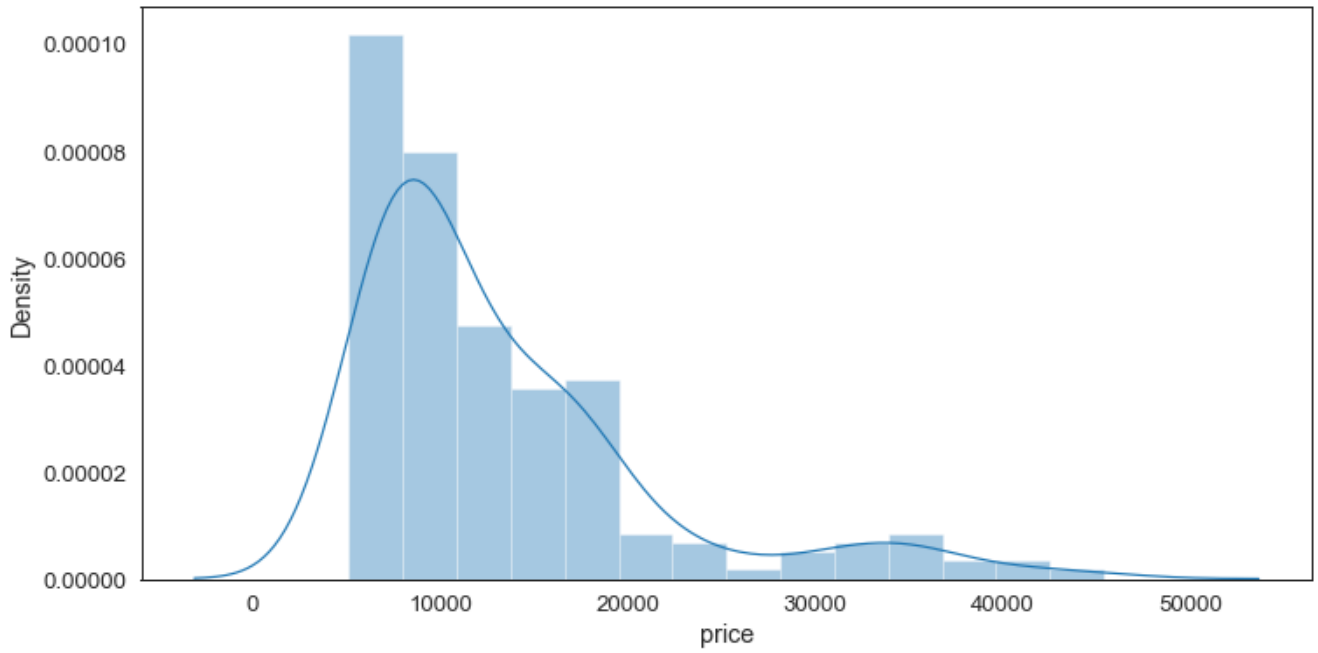
It's supervised learning. Because we have labeled inputs and output data.

## 5. Write a short description about dataset distribution. **PDF**

I want to check the distribution of target and then distribution of all numerical features in dataset:

- DISTRIBUTION OF TARGET

  In below, you can see the target distribution:



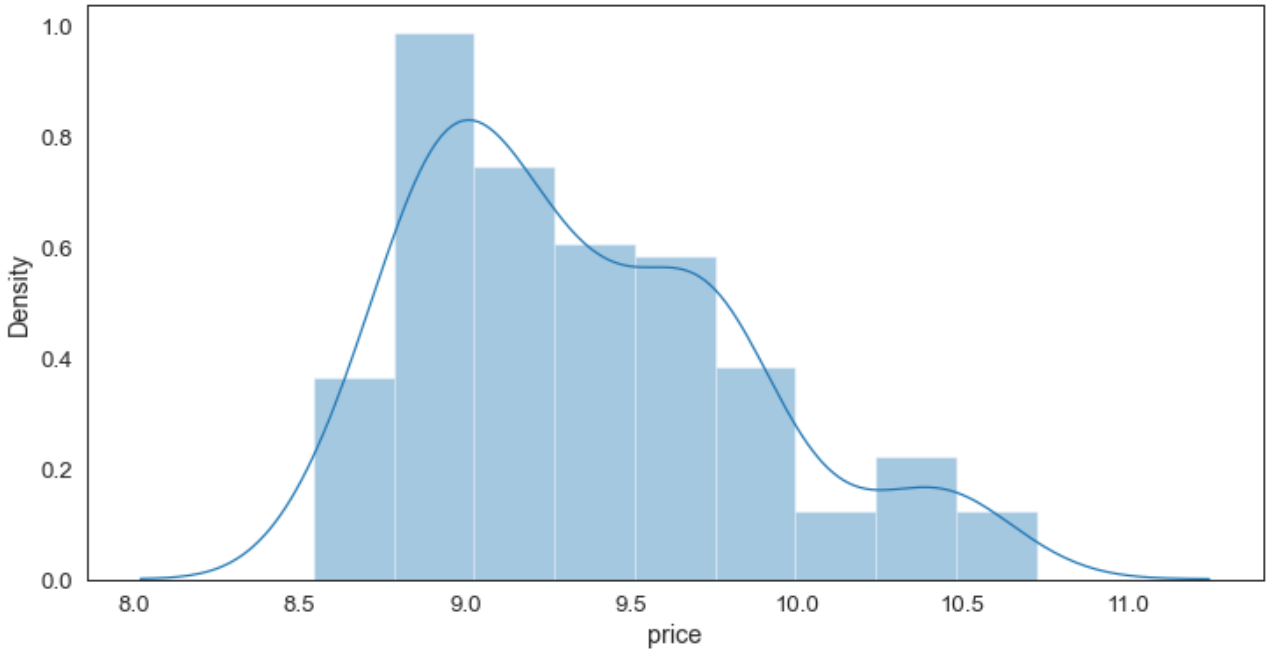

From the above histogram, we can realize that the target has left skewness. So, we should transform it to a normal distribution by removing skewness.

For that, we can use the log-transformation. The log-transformation does remove or reduce skewness.

- Removing the skewness of target ¶

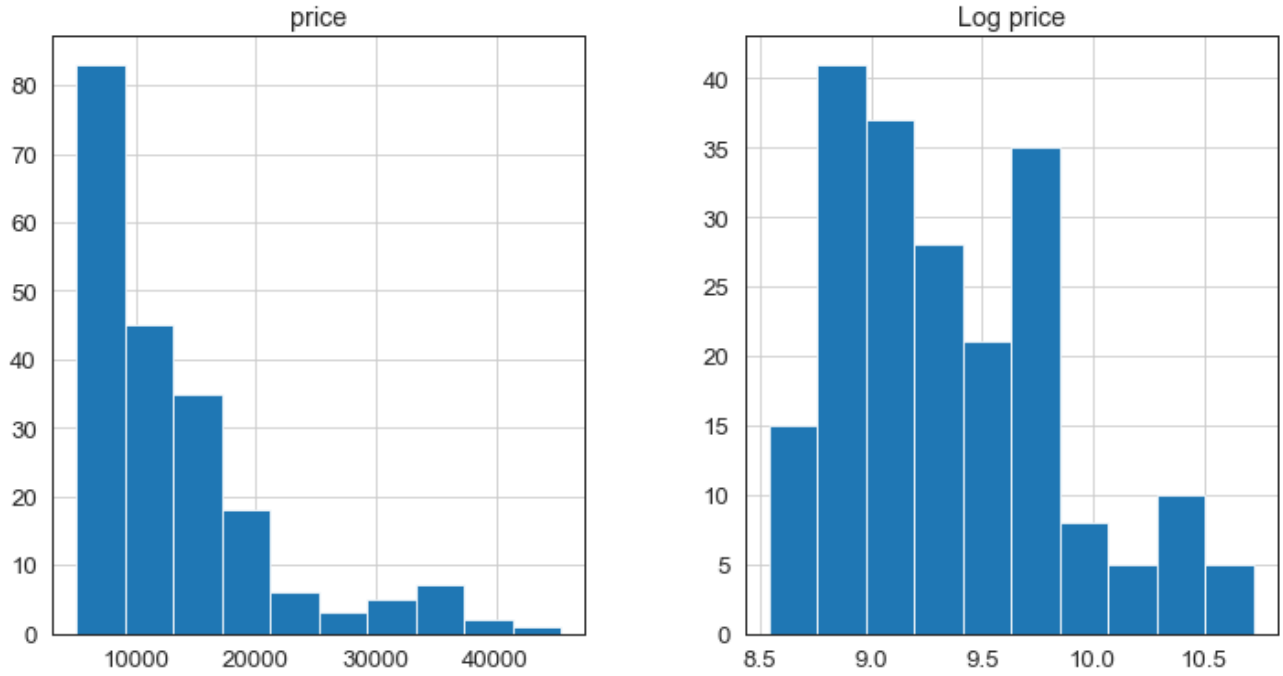
```
target_log = np.log(target)
sns.distplot(target_log, hist=True)
```

And the histogram of target after removing skewness:




Now, target has normal distribution.

Comparing the target before removing skewness and after that:




The left histogram is target distribution: left skewed.

And the right histogram is target_log distribution (after removing skewness): normal distribution.

- ## DISTRIBUTION OF ALL FEATURES

  First, we drop the target (price column) from the data frame and then divide the dataset into numerical data and categorical data in new data frames:

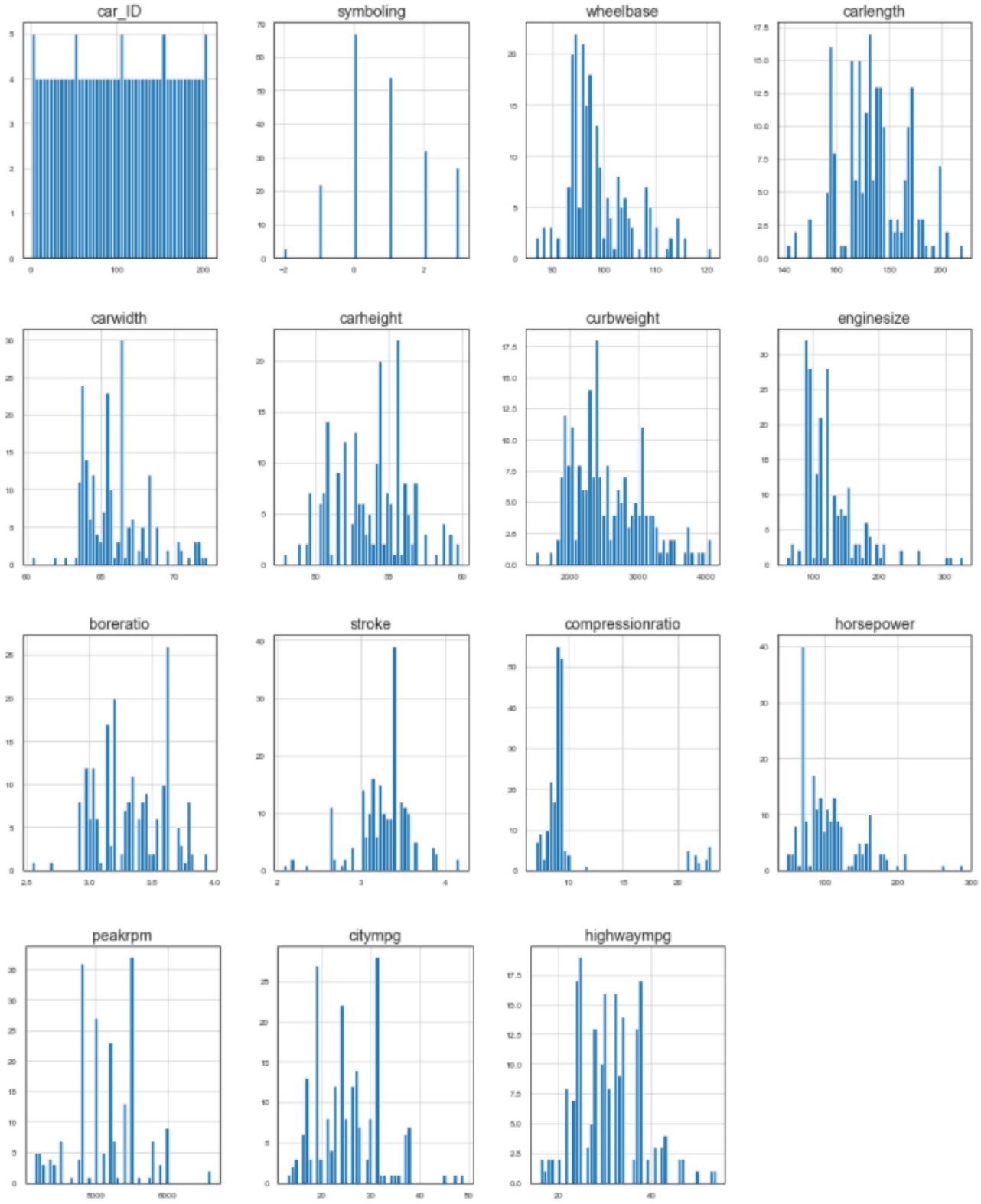
  - Drop target column from main dataframe

```
df = main_df
main_df = main_df.drop(['price'], axis=1)
main_df.head()
```

  - Dividing data to numeric and categorical data:

```
# Save all categorical columns in a list
categorical_columns = [col for col in main_df.columns.values if main_df[col].dtype == 'object']

# dataframe with categorical featurs
df_cat = main_df[categorical_columns]

# dataframe with nomerical featurs
df_num = main_df.drop(categorical_columns, axis=1)
```

Then we get the distribution of all numerical features:



So, there are some features in our data which are left-skewed like wheelbase or enginesize, and also some of features have normal distribution. So, we should perform some transformation on this data to get the normalized data.

So, we can remove the skewness of data using the log-transformation:

- Applying log + 1 transformation for all numeric features with skewness over .75

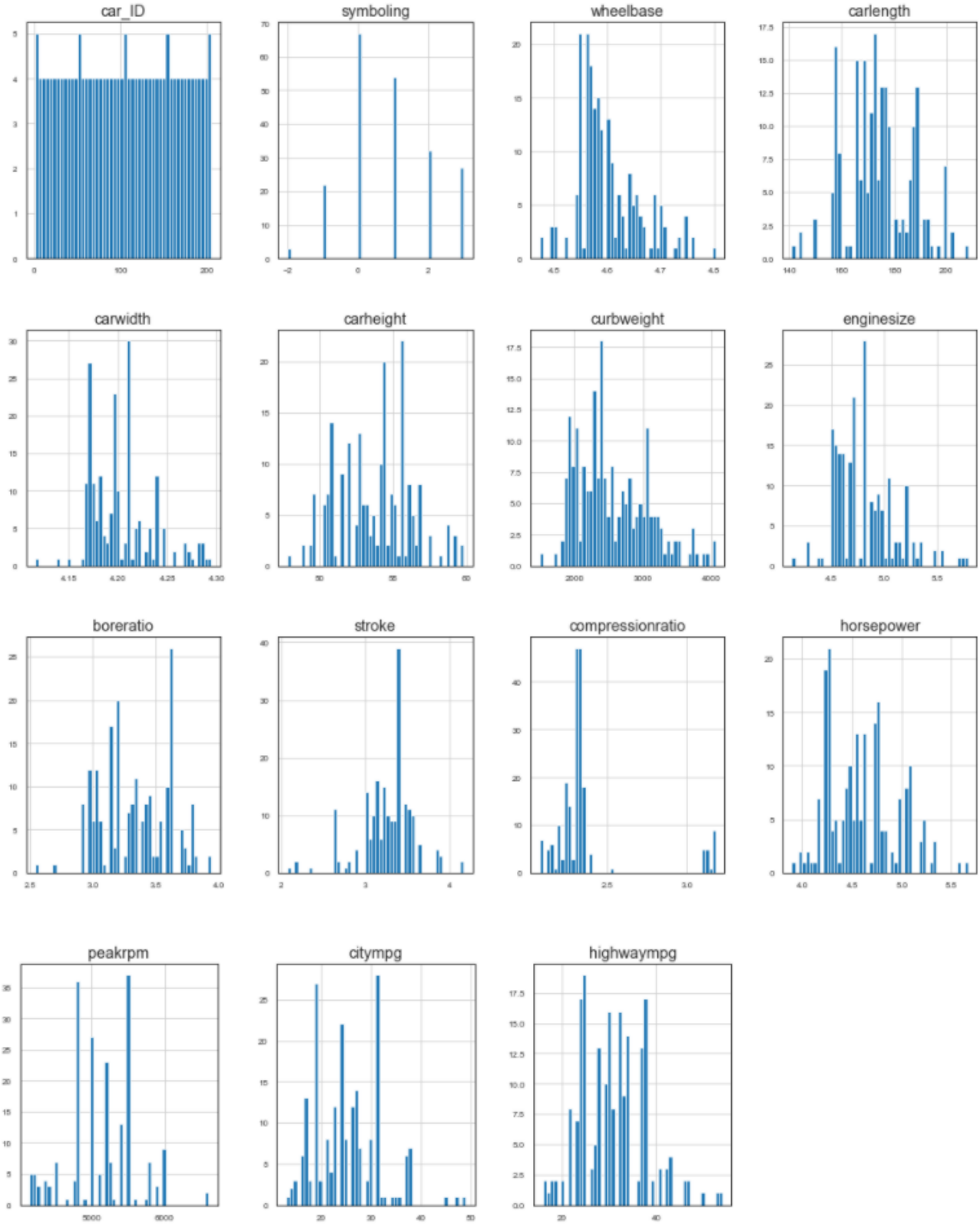
```
df_num_skew = df_num.apply(lambda x: skew(x.dropna()))
df_num_skew = df_num_skew[df_num_skew > .75]

df_num[df_num_skew.index] = np.log1p(df_num[df_num_skew.index])
```

```
# List of variables with skewness over .75
df_num_skew
```

```
wheelbase           1.042514
carwidth            0.897375
enginesize          1.933375
compressionratio    2.591720
horsepower          1.395006
dtype: float64
```

I get the columns from the data frame that have the skewness over .75 and then perform the log + 1 transformation.

So, we have:


car_ID
symboling
wheelbase
carlength
carwidth
carheight
curbweight
enginesize
boreratio
stroke
compressionratio
horsepower
peakrpm
citympg
highwaympg

Now, we have normalized features. For enginesize column, we can see that the data have the normal distribution now.

After removing skewness from all of features and target, the next step is data normalization.

- NORMALIZATION DATA

Normalization usually means to scale a variable to have a value between 0 and 1.

Before normalization data, from describing method for numerical features, we have:

```
# descibe the numeric data
df_num.describe()
```

| | car_ID | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio | stro |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.0000 |
| mean | 103.000000 | 0.834146 | 98.756585 | 174.049268 | 65.907805 | 53.724878 | 2555.565854 | 126.907317 | 3.329756 | 3.2554 |
| std | 59.322565 | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 0.270844 | 0.3135 |
| min | 1.000000 | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 2.540000 | 2.0700 |
| 25% | 52.000000 | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 3.150000 | 3.1100 |
| 50% | 103.000000 | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 3.310000 | 3.2900 |
| 75% | 154.000000 | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 3.580000 | 3.4100 |
| max | 205.000000 | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 3.940000 | 4.1700 |

From this information, we can realize that the values are not between 0 and 1. So, we need to normalize this data.

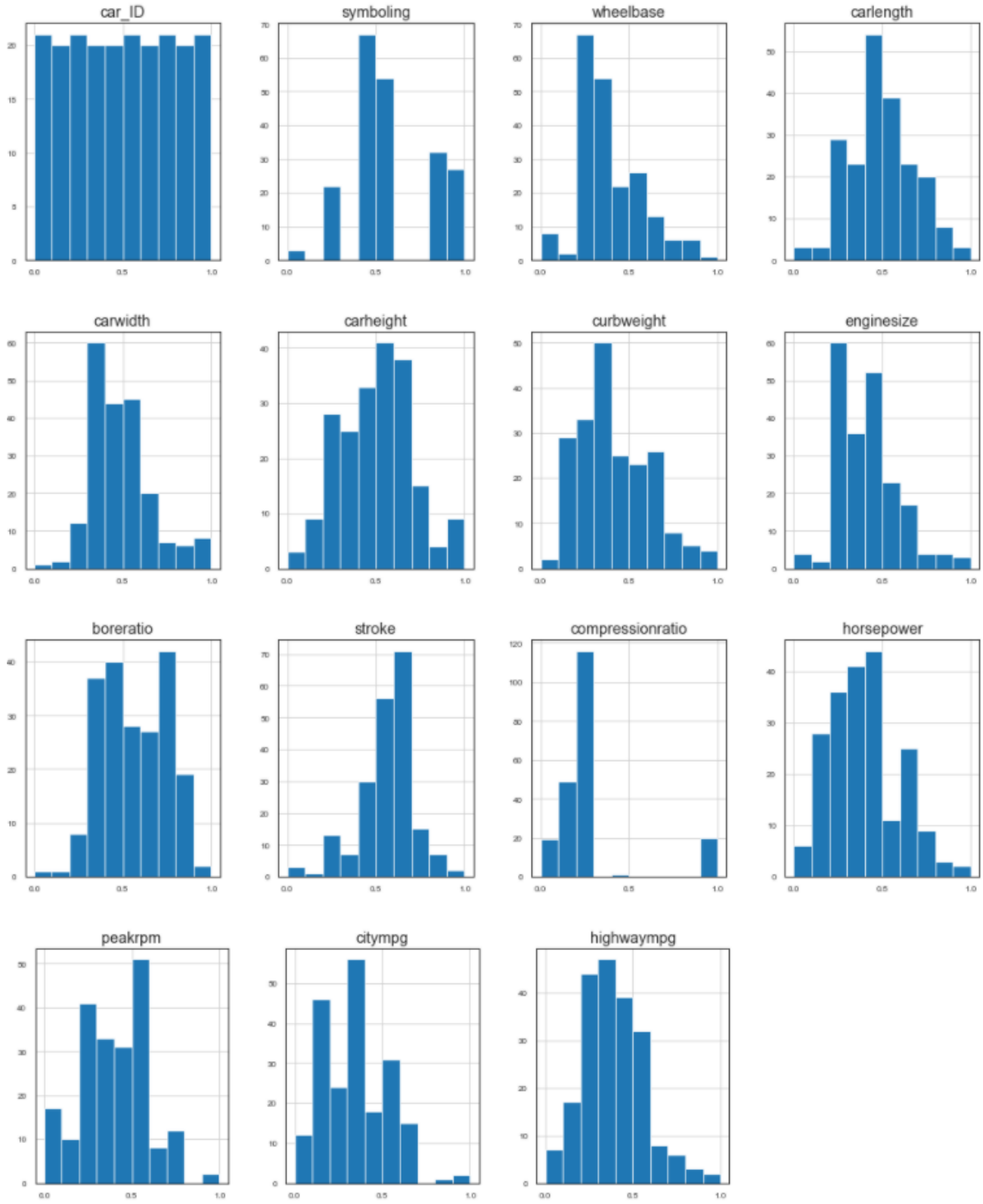
Formula for normalization: x normalized = (x – x_min) / (x_max – x_min)

after normalization:

- Data normalization

```
df_num = ((df_num - df_num.min()) / (df_num.max() - df_num.min()))
df_num.describe()
```

| | car_ID | symboling | wheelbase | carlength | carwidth | carheight | curbweight | enginesize | boreratio | strok |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.00000 |
| mean | 0.500000 | 0.566829 | 0.388000 | 0.491780 | 0.486815 | 0.493740 | 0.414106 | 0.409874 | 0.564111 | 0.56448 |
| std | 0.290797 | 0.249061 | 0.177901 | 0.184139 | 0.177020 | 0.203627 | 0.201971 | 0.168886 | 0.193460 | 0.14933 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| 25% | 0.250000 | 0.400000 | 0.261320 | 0.376119 | 0.336416 | 0.350000 | 0.254849 | 0.275334 | 0.435714 | 0.49523 |
| 50% | 0.500000 | 0.600000 | 0.339527 | 0.479104 | 0.455430 | 0.525000 | 0.359193 | 0.402120 | 0.550000 | 0.58095 |
| 75% | 0.750000 | 0.800000 | 0.501858 | 0.626866 | 0.571964 | 0.641667 | 0.561288 | 0.498364 | 0.742857 | 0.63809 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |


car_ID
symboling
wheelbase
carlength
carwidth
carheight
curbweight
enginesize
boreratio
stroke
compressionratio
horsepower
peakrpm
citympg
highwaympg

## 10. How much is this model accurate (Use SKlearn library)? **PDF**

we can calculate the accuracy of regression model by calculating the MSE and RMSE:

### 10. How much is this model accurate ( Use SKlearn library )? PDF

```
from sklearn.metrics import mean_squared_error
import math

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2_score = sgdr.score(X_test,y_test)

print('RMSE: ', rmse)
print('R²:', r2_score)
```

```
RMSE:  0.2843583732401856
R²: 0.7167468581078008
```