Car Price Prediction

Problem Description:

- The target variable is the price of the car which is a continuous variable.
- The people who are going to buy a new automobile for their constant activities will be having these types of problem.
- Using advanced machine learning techniques appropriate for statistical analysis, we can help solve the challenge mentioned above. This project aims to build a model that predicts the automobile price make and model-given attributes of the automobile.

Data Set:

- To Build such a model, we take the data from the source which is the Dataset file: 'cars_class.csv'
- This is a regression data set.
- The data set has 206 samples.
- There are 25 features.
- The target variable is the price of the car

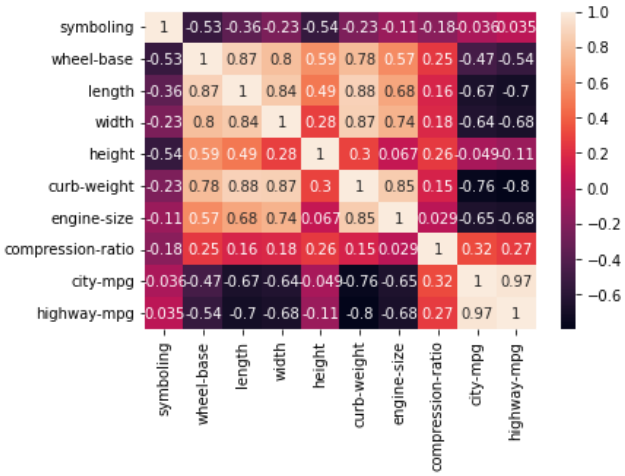
Feature Engineering:

- The features of the data are as follows:
- symbolling
- normalized-losses
- make
- fuel-type aspiration
- num-of-doors
- body-style
- drive-wheels
- engine-location
- wheel-base

- length
- width
- height
- curb-weight
- engine-type
- num-of-cylinders
- engine-size
- fuel-system
- bore stroke
- compression-ratio
- horsepower
- peak-rpm
- city-mpg highway-mpg

The Feature ‘make’ will not be useful in the data frame as we are going to predict the price.

- I have used the correlation between the features to delete some features to get the better results.
- correlation = data.corr()
- sns.heatmap(correlation, xticklabels= correlation.columns, yticklabels= correlation.columns, annot= True)



- The dropped features in the data are:

- 'highway-mpg’, ‘curb-weight’, ‘length', 'make'.

Data Preprocessing:

- There are no missing values by checking with the null function.
- But the dataset is having missing values in the form of “?”.
- The feature ‘engine-type’ has a missing value of ‘l’.
- And the datatypes of our data are:
- **s**ymbolling int64
- normalized-losses object
- fuel-type object
- aspiration object
- num-of-doors object
- body-style object
- drive-wheels object
- engine-location object
- wheel-base float64
- width float64
- height float64
- engine-type object
- num-of-cylinders object
- engine-size int64
- fuel-system object
- bore object
- stroke object
- compression-ratio float64
- horsepower object
- peak-rpm object
- city-mpg int64

The data set has been into two parts and that is 80% of training data and 20% of test data.

- The shape of the training data is (164,21).
- I see that the features ‘num-of-doors’ and ‘num-of-cylinders’ are in word format which should be in number format.
- for i in X_train['num-of-cylinders']:
- print(w2n.word_to_num(i))
- X_train['num-of-cylinders'] = w2n.word_to_num(i)
- a= []
- for i in X_train['num-of-doors']:
- w2n.word_to_num(i)
- a.append(w2n.word_to_num(i))
- print(a)

I see that some of the features in the data should be numeric but they are in the object so I have changed the features to a numeric format.

- X_train['horsepower'] = pd.to_numeric(X_train['horsepower'], errors= 'coerce')
- X_train['peak-rpm'] = pd.to_numeric(X_train['peak-rpm'], errors= 'coerce')
- X_train['bore'] = pd.to_numeric(X_train['bore'], errors= 'coerce')
- X_train['stroke'] = pd.to_numeric(X_train['stroke'], errors= 'coerce')
- X_train['normalized-losses'] = pd.to_numeric(X_train['normalized-losses'], errors= 'coerce')
- The feature’s missing values are replaced by the replace() function.
- X_train[['bore', 'stroke']] = X_train[['bore', 'stroke']].replace('?', np.nan)
- X_train['engine-type'] = X_train['engine-type'].replace('l', np.nan)

The numerical features which are having missing values are replaced by SimpleImputer(mean).

The Categorical features which are having missing values are replaced by SimpleImputer(mostfrequent).

- After some processing the datatypes of the data is:
- symboling int64
- normalized-losses float64

- fuel-type object
- aspiration object
- num-of-doors int64
- body-style object
- drive-wheels object
- engine-location object
- wheel-base float64
- width float64
- height float64
- engine-type object
- num-of-cylinders int64
- engine-size int64
- fuel-system object
- bore float64
- stroke float64
- compression-ratio float64
- horsepower float64
- peak-rpm float64
- city-mpg int64

The categorical features are now transformed to numerical features using OneHotEncoder.

After OneHotEncoding the shape of the dataset is (164,41).

As the features are very high, I have used Principal Component Analysis on the features for the feature reduction.

- pca = PCA(n_components= 3, random_state= 1)
- pca.explained_variance_ratio_
- [0.00364811, 0.01429635, 0.98088454]
- So, now the features are three for the dataset.

Data Exploratory:

- The first five rows of the data are:

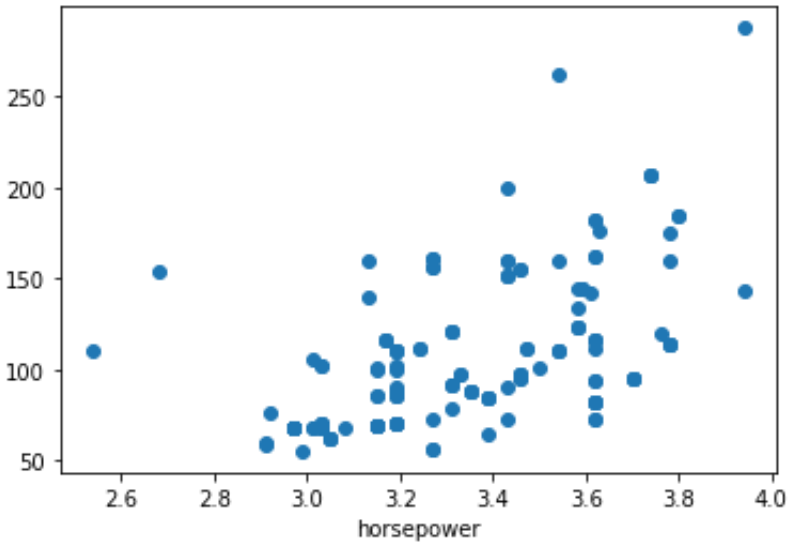
	0	1	2
0	96.709201	89.469882	97.341561
1	-305.025975	-54.629323	19.793724
2	94.751572	-27.564535	-29.907533
3	144.643602	-32.451335	-24.943983
4	95.201358	-47.809833	12.376206

- The shape of the dataset is (164,3).

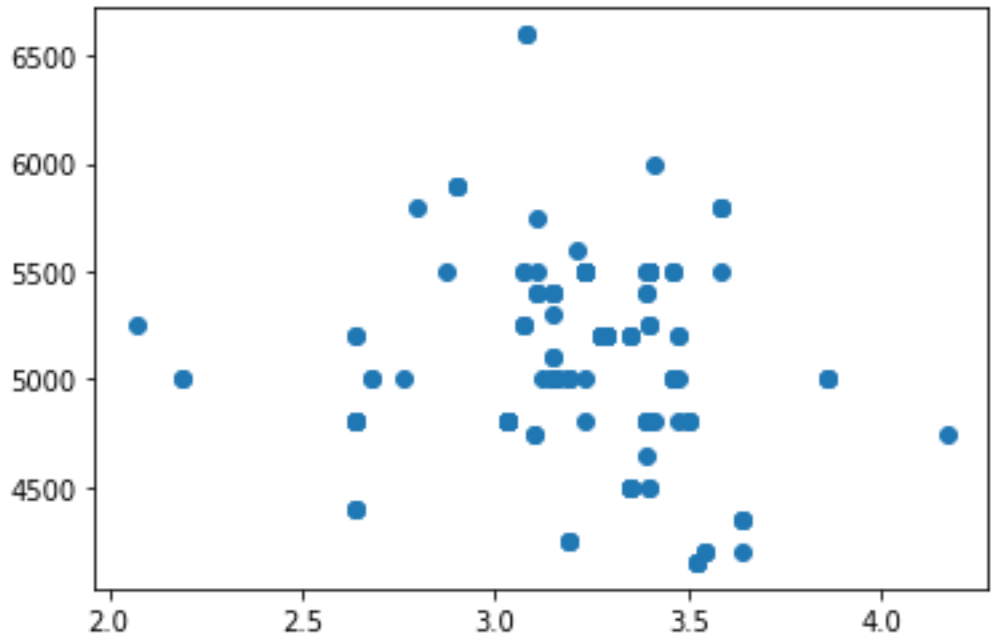
- The Summary of the statistics values in the data are:

	0	1	2
count	164.00	164.00	164.00
mean	0.00	0.00	0.00
std	471.15	56.88	28.73
min	-955.21	-70.90	-53.25
25%	-305.08	-45.15	-20.62
50%	94.55	-16.75	-4.59
75%	394.83	29.95	15.96
max	1495.88	247.65	97.34

- The relation between the bore and horsepower features is:



The relation of stroke and peak-rpm feature as follows:



The features are now scaled with same mean and variance using StandardScaler.

Data Modelling:

As the target is a continuous variable, we will be using Regression Machine learning techniques.

Now the data has been cleaned and ready for the modelling, we can apply ML techniques.

I have used some number of regression ML techniques and they are:

- LinearRegression
- SGDRegressor
- Ridge
- Lasso
- ElasticNet
- KNeighborsRegressor
- DecisionTreeRegressor
- RandomForestRegressor
- GradientBoostingRegressor

The R-2 scores of the training data for the above techniques are:

- 0.7331948103334043
- 0.73318600865133
- 0.7331678793945491
- 0.7331947627558772
- 0.6517009088218485
- 0.8390425923748264
- 0.9956934760200945
- 0.9752248333694986
- 0.9771739064973282

After the scores, I have gone ahead with the DecisionTreeRegressor, RandomForestRegressor and GradientBoostingRegressor for the Hyper-Parameter Tuning.

- After Tuning the test data has been transformed as per the preprocessing of the training data.

- Now, The testing data will be transformed as per the finalmodel.
- The final model is RandomForestRegressor after the tuning.
- final_model = RandomForestRegressor(n_estimators=100, random_state=1)
- final_model.fit(X_train, y_train)
- final_model.score(X_test, y_test)
- The R-2 score of the test data is [0.7662106321763562].
- The importance of the features after modelling the data is:
- [0.10256309, 0.82092912, 0.07650779].