Vehicle Price Prediction:-

In [120…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [69]:
```
vpp=pd.read_csv(r"C:\Users\Vicky Yewle\Downloads\Machine Learning\Linear Regression\Veh
vpp.head()
```

Out[69]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	name	Kilometer_per_lite
0	1	8	307.0	130	3504	12.0	1970	chevrolet chevelle malibu	7.65258
1	1	8	350.0	165	3693	11.5	1970	buick skylark 320	6.37715
2	1	8	318.0	150	3436	11.0	1970	plymouth satellite	7.65258
3	1	8	304.0	150	3433	12.0	1970	amc rebel sst	6.80229
4	1	8	302.0	140	3449	10.5	1970	ford torino	7.22744

In [70]:
```
vpp.shape
```

Out[70]:
```
(398, 9)
```

In [71]:
```
vpp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   origin              398 non-null    int64  
 1   cylinders           398 non-null    int64  
 2   displacement        398 non-null    float64
 3   horsepower          398 non-null    object 
 4   weight              398 non-null    int64  
 5   acceleration        398 non-null    float64
 6   year                398 non-null    int64  
 7   name                398 non-null    object 
 8   Kilometer_per_liter  398 non-null    float64
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

In [72]:
```
#It seems that there is ? value in the column that confirms that
```

```
flagged_value = vpp['horsepower']== '?'
flagged_value.value_counts()
```

Out[72]:
```
False    392
True       6
Name: horsepower, dtype: int64
```

In [73]:
```
vpp['horsepower'] = vpp['horsepower'].replace('?',np.nan)
vpp["horsepower"] =vpp["horsepower"].apply(pd.to_numeric)
vpp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   origin               398 non-null    int64  
 1   cylinders            398 non-null    int64  
 2   displacement         398 non-null    float64
 3   horsepower           392 non-null    float64
 4   weight               398 non-null    int64  
 5   acceleration         398 non-null    float64
 6   year                 398 non-null    int64  
 7   name                 398 non-null    object 
 8   Kilometer_per_liter  398 non-null    float64
dtypes: float64(4), int64(4), object(1)
memory usage: 28.1+ KB
```

In [74]:
```
del vpp['name']
```

In [75]:
```
vpp
```

Out[75]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	Kilometer_per_liter
0	1	8	307.0	130.0	3504	12.0	1970	7.652587
1	1	8	350.0	165.0	3693	11.5	1970	6.377156
2	1	8	318.0	150.0	3436	11.0	1970	7.652587
3	1	8	304.0	150.0	3433	12.0	1970	6.802299
4	1	8	302.0	140.0	3449	10.5	1970	7.227443
...	...	...	...	...	...	...	...	...
393	1	4	140.0	86.0	2790	15.6	1982	11.478880
394	2	4	97.0	52.0	2130	24.6	1982	18.706323
395	1	4	135.0	84.0	2295	11.6	1982	13.604599
396	1	4	120.0	79.0	2625	18.6	1982	11.904024
397	1	4	119.0	82.0	2720	19.4	1982	13.179455

398 rows × 8 columns

In [76]:
```
#DataFrameName.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

vpp.dropna(subset =['horsepower'], inplace=True)
vpp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   origin               392 non-null    int64  
 1   cylinders            392 non-null    int64  
 2   displacement         392 non-null    float64
 3   horsepower           392 non-null    float64
 4   weight               392 non-null    int64  
 5   acceleration         392 non-null    float64
 6   year                 392 non-null    int64  
 7   Kilometer_per_liter  392 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 27.6 KB
```

In [77]:
```
#
vpp['year']
```

Out[77]:
```
0      1970
1      1970
2      1970
3      1970
4      1970
       ...
393    1982
394    1982
395    1982
396    1982
397    1982
Name: year, Length: 392, dtype: int64
```

In [78]:
```
# To scale data into dataset
vpp['year']=vpp['year']-vpp['year'].min()
vpp['year'].describe()
```

Out[78]:
```
count    392.000000
mean       5.979592
std        3.683737
min        0.000000
25%        3.000000
50%        6.000000
75%        9.000000
max       12.000000
Name: year, dtype: float64
```

In [79]:
```
vpp.describe()
# this tells that origin is not gradually increasing it is constant till 50% and 75%
```

Out[79]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	Kilome
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	
mean	1.576531	5.471939	194.411990	104.469388	2977.584184	15.541327	5.979592	
std	0.805518	1.705783	104.644004	38.491160	849.402560	2.758864	3.683737	
min	1.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	0.000000	
25%	1.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	3.000000	
50%	1.000000	4.000000	151.000000	93.500000	2803.500000	15.500000	6.000000	
75%	2.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	9.000000	

	origin	cylinders	displacement	horsepower	weight	acceleration	year	Kilome
max	3.000000	8.000000	455.000000	230.000000	5140.000000	24.800000	12.000000	

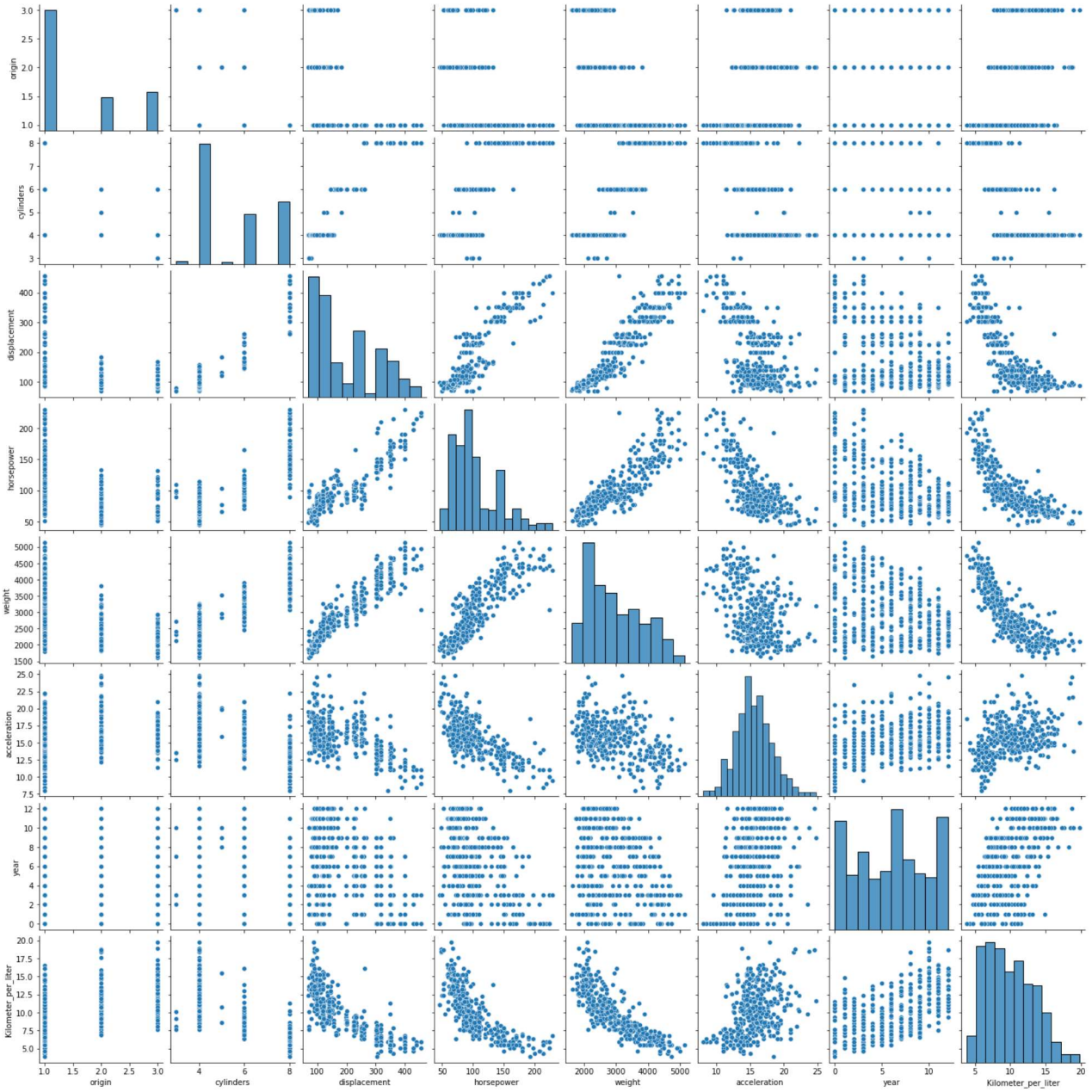
In [80]:
```
vpp['horsepower'].value_counts()
print('there is',round((6/398)*100,2),'% NA data which can be deleted')
```

```
there is 1.51 % NA data which can be deleted
```

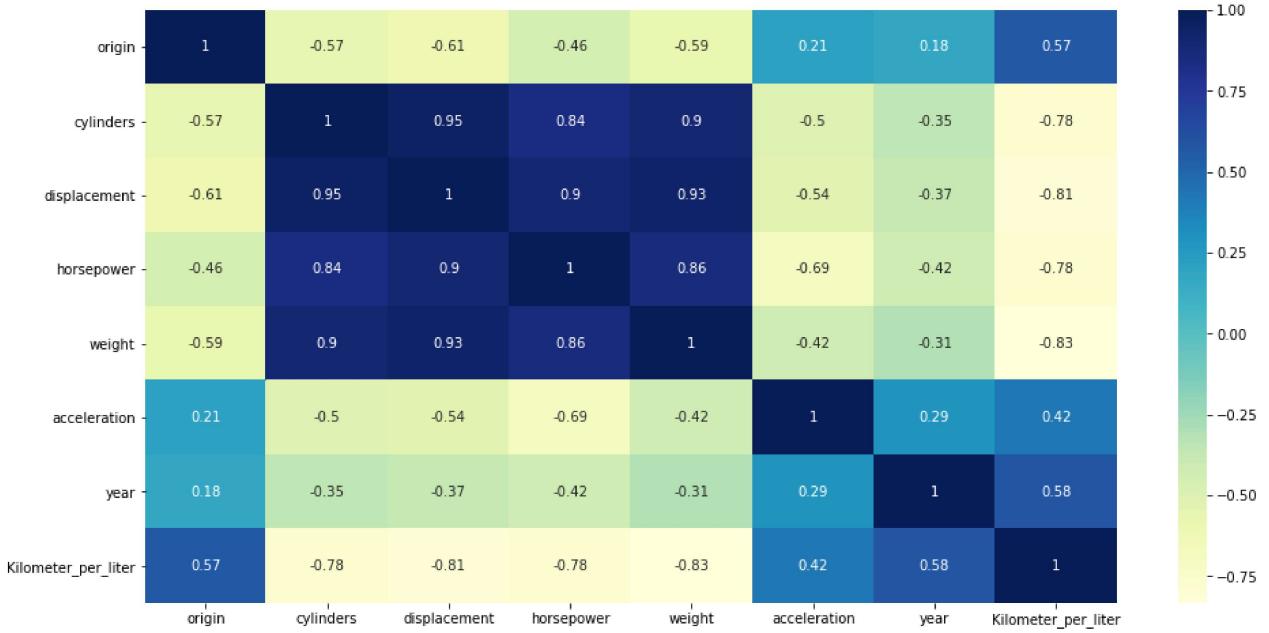
In [81]:
```
#All NA's are removed so our data is now clean.Hence we will first visualize the datase
#We will plot pairplot for the variation of 2 attributes
```

In [82]:
```
#Pairplot
plt.figure(figsize=(20, 10))
sns.pairplot(vpp)
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```



In [83]:
```
#Plotting Heatmap and Calculating correlations of variable
# figure size
plt.figure(figsize=(16,8))

# heatmap
sns.heatmap(vpp.corr(), cmap="YlGnBu", annot=True)
plt.show()
```



In [84]:
```
#As from origin vs origin graph we got that there are variable repeating in ORIGIN colu
#Now we will create dummy variables for origin because there are  entities mentioned in

vpp['origin'] = vpp['origin'].astype(str)
vpp_dummies = pd.get_dummies(vpp['origin'], drop_first=True)
vpp_dummies.head()
```

Out[84]:

	2	3
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

In [85]:
```
vpp_dummies.columns = ['origin_two','origin_three']
vpp_dummies.columns
```

Out[85]:
```
Index(['origin_two', 'origin_three'], dtype='object')
```

In [86]:
```
vpp=pd.concat([vpp_dummies,vpp], axis=1)
del vpp['origin']
vpp
```

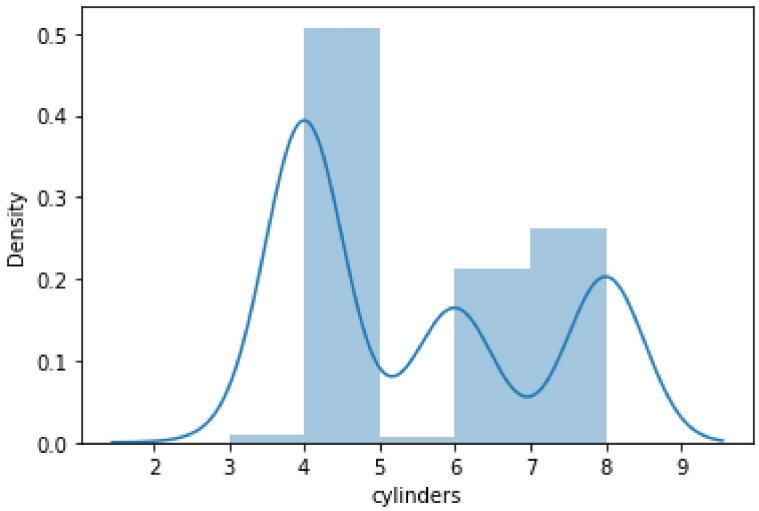
Out[86]:

	origin_two	origin_three	cylinders	displacement	horsepower	weight	acceleration	year	Kilomet
0	0	0	8	307.0	130.0	3504	12.0	0	

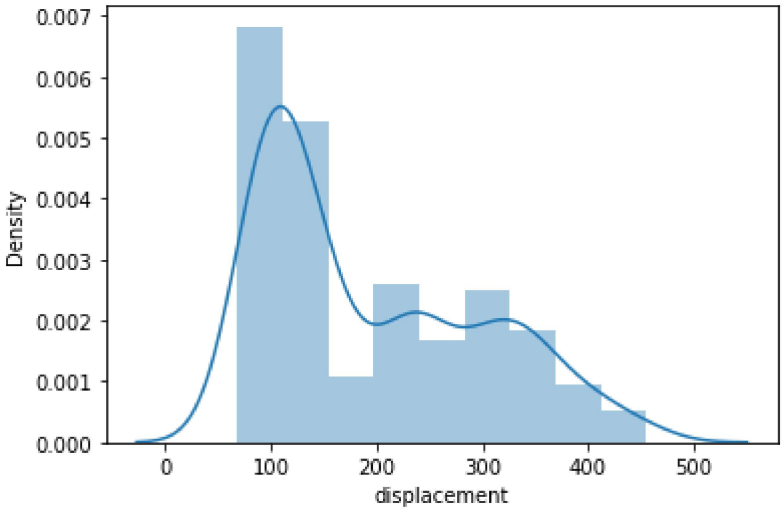
	origin_two	origin_three	cylinders	displacement	horsepower	weight	acceleration	year	Kilomet
1	0	0	8	350.0	165.0	3693	11.5	0	
2	0	0	8	318.0	150.0	3436	11.0	0	
3	0	0	8	304.0	150.0	3433	12.0	0	
4	0	0	8	302.0	140.0	3449	10.5	0	
...	...	...	...	...	...	...	...	...	
393	0	0	4	140.0	86.0	2790	15.6	12	
394	1	0	4	97.0	52.0	2130	24.6	12	
395	0	0	4	135.0	84.0	2295	11.6	12	
396	0	0	4	120.0	79.0	2625	18.6	12	
397	0	0	4	119.0	82.0	2720	19.4	12	

392 rows × 9 columns

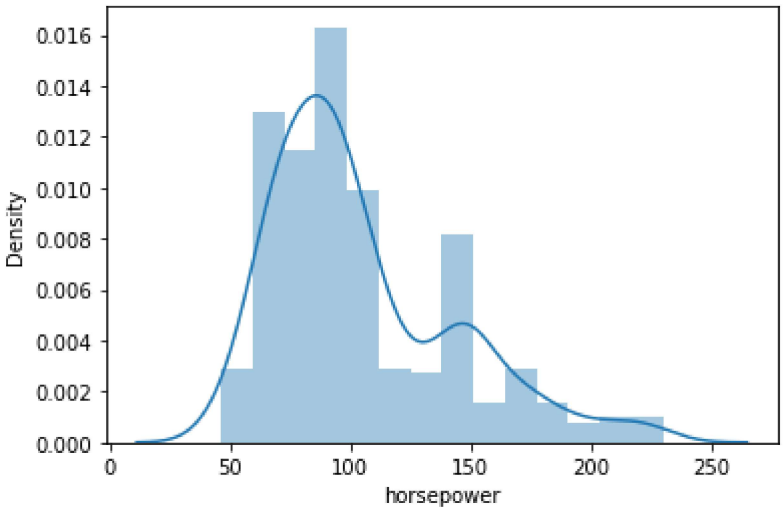
In [87]:
```
# now we will see the distribution of various datapoints

sns.distplot(vpp['cylinders'])
plt.show()
```



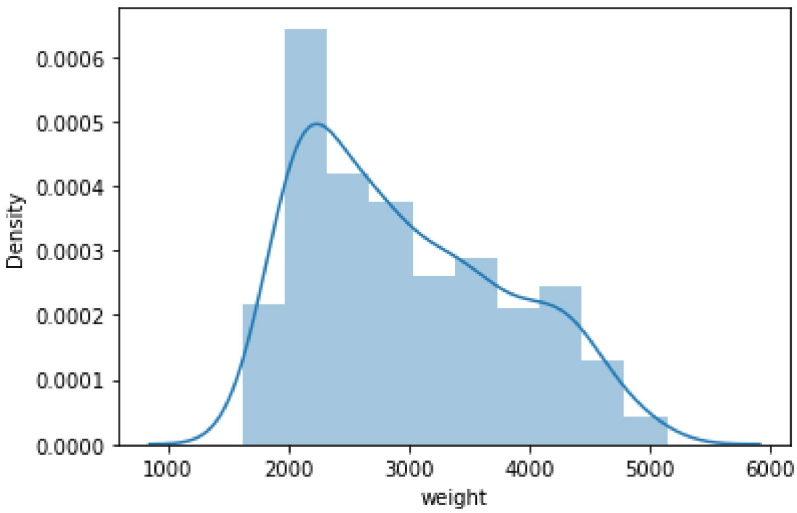
In [88]:
```
sns.distplot(vpp['displacement'])
plt.show()
```



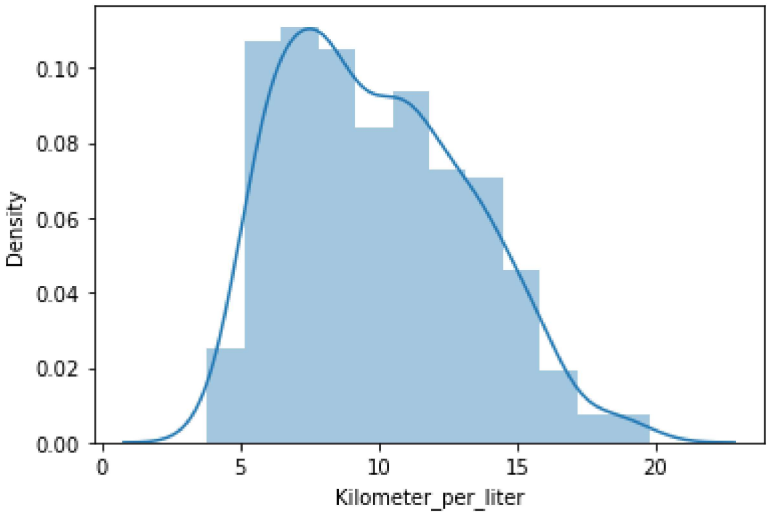
In [89]:
```
sns.distplot(vpp['horsepower'])
plt.show()
```



In [90]:
```
sns.distplot(vpp['weight'])
plt.show()
```



In [91]:
```
sns.distplot(vpp['Kilometer_per_liter'])
plt.show()
```



In [94]:
```
# scaling the features
#from sklearn.preprocessing import scale
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()


# storing column names in cols, since column names are (annoyingly) lost after
# scaling (the df is converted to a numpy array)

#Subsetting data
cols = vpp[['cylinders','displacement','horsepower','weight','acceleration','year']]

X = pd.DataFrame(scaler.fit_transform(cols))
X
```

Out[94]:

	0	1	2	3	4	5
0	1.0	0.617571	0.456522	0.536150	0.238095	0.0
1	1.0	0.728682	0.646739	0.589736	0.208333	0.0
2	1.0	0.645995	0.565217	0.516870	0.178571	0.0
3	1.0	0.609819	0.565217	0.516019	0.238095	0.0
4	1.0	0.604651	0.510870	0.520556	0.148810	0.0
...	...	...	...	...	...	...
387	0.2	0.186047	0.217391	0.333711	0.452381	1.0
388	0.2	0.074935	0.032609	0.146583	0.988095	1.0
389	0.2	0.173127	0.206522	0.193365	0.214286	1.0
390	0.2	0.134367	0.179348	0.286929	0.630952	1.0
391	0.2	0.131783	0.195652	0.313864	0.678571	1.0

392 rows × 6 columns

In [95]:
```
#Renaming Dataframe
X.columns = list(cols.columns)
```

```
#Testing
scaled_data = pd.concat([vpp_dummies,X,], axis=1)
scaled_data.head()
```

Out[95]:

	origin_two	origin_three	cylinders	displacement	horsepower	weight	acceleration	year
0	0.0	0.0	1.0	0.617571	0.456522	0.536150	0.238095	0.0
1	0.0	0.0	1.0	0.728682	0.646739	0.589736	0.208333	0.0
2	0.0	0.0	1.0	0.645995	0.565217	0.516870	0.178571	0.0
3	0.0	0.0	1.0	0.609819	0.565217	0.516019	0.238095	0.0
4	0.0	0.0	1.0	0.604651	0.510870	0.520556	0.148810	0.0

In [96]:
```
y = vpp['Kilometer_per_liter']
```

In [97]:
```
#Checking if the number of rows are same
len(X) == len(y)
```

Out[97]: True

In [98]:
```
# split into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.6,
                                                    test_size = 0.4, random_state=100)
```

In [99]:
```
#Checking if the number of rows and columns are same
print(len(X_test) == len(y_test))
print(len(X_train) == len(y_train))
```

```
True
True
```

In [100…
```
from sklearn import linear_model, metrics
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
```

In [101…
```
# list of alphas to tune
params = {'alpha': [0.01,0.20,0.100,4,5,10]}


ridge = Ridge()

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.4s finished
```

Out[101…

```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [0.01, 0.2, 0.1, 4, 5, 10]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [102…

```
cv_results=pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

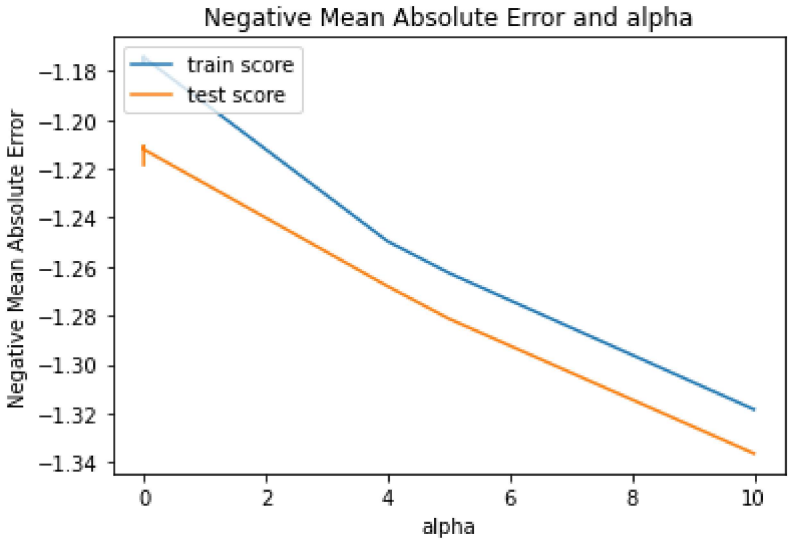
Out[102…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score
0	0.008401	0.001744	0.005200	0.001328	0.01	{'alpha': 0.01}	-1.341819
1	0.005800	0.000748	0.003800	0.001167	0.2	{'alpha': 0.2}	-1.355283
2	0.005632	0.000867	0.003567	0.000787	0.1	{'alpha': 0.1}	-1.348474
3	0.007602	0.001502	0.003997	0.000638	4	{'alpha': 4}	-1.425950
4	0.005203	0.000748	0.003600	0.000490	5	{'alpha': 5}	-1.432332

5 rows × 21 columns

In [103…

```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')
plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [104…
```
alpha = 5 # till 5 the difference between train and test is getting smaller and after 5
ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
ridge.coef_
```

Out[104…
```
array([-1.77466747, -1.65106475, -1.85908961, -3.54469828,  0.38186933,
        3.04599534])
```

In [105…
```
#

lasso = Lasso()

params = {'alpha': [0.000000001,0.000000010,0.000000100, 0.000001000,0.000010000,0.0001

# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    0.5s finished
```

Out[105…
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [1e-09, 1e-08, 1e-07, 1e-06, 1e-05, 0.0001,
                                   0.001, 0.01, 0.1, 0.5]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [106…
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

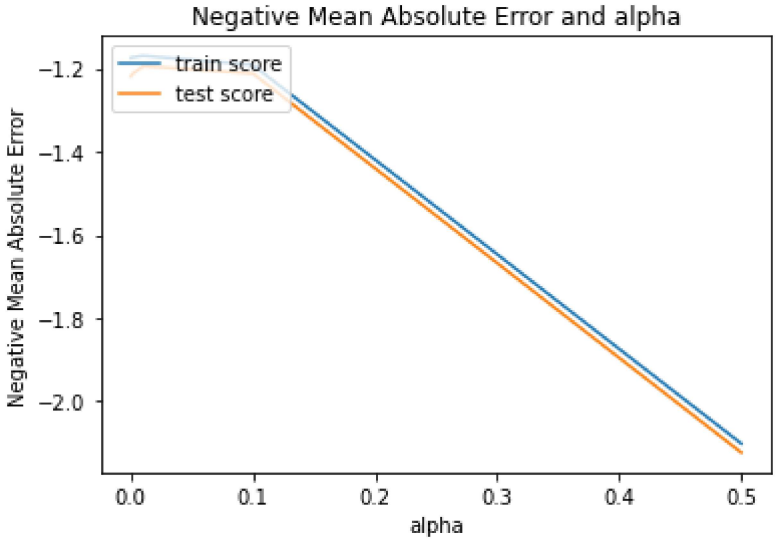
Out[106…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score
0	0.008599	0.002155	0.005601	0.001624	1e-09	{'alpha': 1e-09}	-1.340978
1	0.004600	0.000800	0.003201	0.000400	1e-08	{'alpha': 1e-08}	-1.340978
2	0.004601	0.000801	0.002600	0.000490	1e-07	{'alpha': 1e-07}	-1.340978
3	0.006401	0.002245	0.004200	0.001168	1e-06	{'alpha': 1e-06}	-1.340976
4	0.004399	0.000490	0.003401	0.000491	1e-05	{'alpha': 1e-05}	-1.340957

5 rows × 21 columns

In [107…
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')
```

```
# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [108…
```
alpha = 0.1 # We are using 0.1 as at that point both the train test variance is less an

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[108… `Lasso(alpha=0.1)`

In [109…
```
lasso.coef_
```

Out[109…
```
array([-1.57267151, -0.        , -0.        , -6.65885165,  0.        ,
        3.117601  ])
```

Elastic Net

In [110…
```
elasticnet = ElasticNet()

# cross validation
model_cv = GridSearchCV(estimator = elasticnet,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    1.0s finished
```

Out[110… `GridSearchCV(cv=5, estimator=ElasticNet(),`

```
                      param_grid={'alpha': [1e-09, 1e-08, 1e-07, 1e-06, 1e-05, 0.0001,
                                            0.001, 0.01, 0.1, 0.5]},
                      return_train_score=True, scoring='neg_mean_absolute_error',
                      verbose=1)
```

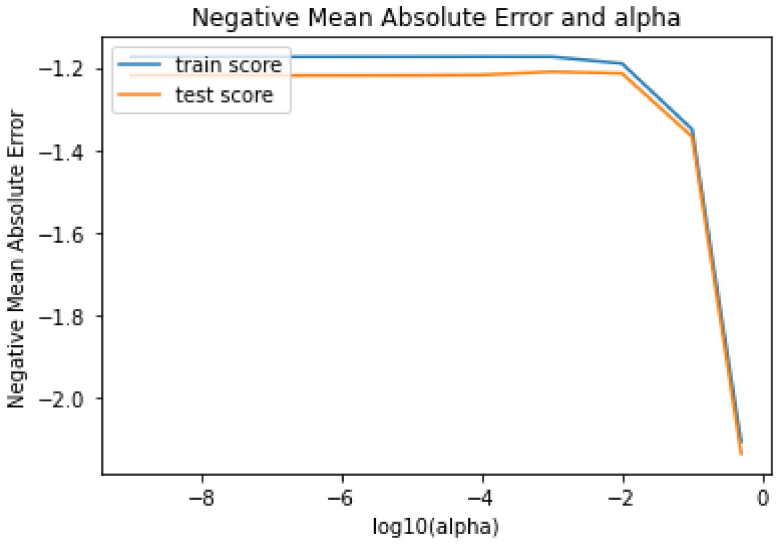
In [111…
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

Out[111…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score
0	0.007599	0.001856	0.003799	0.001164	1e-09	{'alpha': 1e-09}	-1.340978
1	0.004600	0.000490	0.002229	0.000390	1e-08	{'alpha': 1e-08}	-1.340978
2	0.004600	0.000490	0.003200	0.000403	1e-07	{'alpha': 1e-07}	-1.340979
3	0.005197	0.000753	0.004200	0.000746	1e-06	{'alpha': 1e-06}	-1.340985
4	0.005198	0.000407	0.003202	0.000393	1e-05	{'alpha': 1e-05}	-1.341048

5 rows × 21 columns

In [112…
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.plot(np.log10(cv_results['param_alpha']), cv_results['mean_train_score'])
plt.plot(np.log10(cv_results['param_alpha']), cv_results['mean_test_score'])
plt.xlabel('log10(alpha)')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [113…
```
alpha = -0.5
```

```
elasticnet = ElasticNet(alpha=alpha)

elasticnet.fit(X_train, y_train)
```

Out[113… `ElasticNet(alpha=-0.5)`

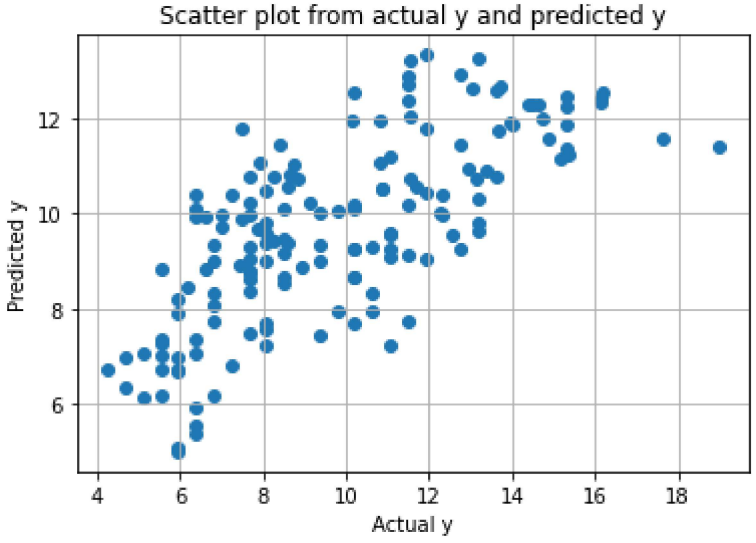
In [114…
```
elasticnet.coef_
```

Out[114…
```
array([ 1.00974188e+289,  9.42731480e+288,  7.59311872e+288,
        9.83752159e+288, -4.48639779e+288, -7.52382544e+288])
```

Stochastic Gredient Descent

In [115…
```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [116…
```
n_iter=10
clf_ = SGDRegressor(max_iter=n_iter)
clf_.fit(X_train, y_train)
y_pred_sksgd=clf_.predict(X_test)
plt.scatter(y_test,y_pred_sksgd)
plt.grid()
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('Scatter plot from actual y and predicted y')
plt.show()

print('Mean Squared Error :',mean_squared_error(y_test, y_pred_sksgd))
print('Mean Absolute Error :',mean_absolute_error(y_test, y_pred_sksgd))
```




```
Mean Squared Error : 4.827343774985655
Mean Absolute Error : 1.8310778639273644
```

In [117…
```
def calculate_aic(n, mse, num_params):
    aic = n * np.log(mse) + 2 * num_params
    return aic

def calculate_bic(n, mse, num_params):
    bic = n * np.log(mse) + np.log(n) * num_params
    return bic
```

In [118…

```
# proportion by which dependent variable can relate with independent variable.

features = X_train.columns

regression = LinearRegression()

selected_features = []
min_aic = np.inf
for step in range(0, 20, 2):
    for feature in features:
        testing_features = selected_features + [feature]
        regression.fit(X_train[testing_features], y_train)
        y_pred = regression.predict(X_test[testing_features])
        mse = metrics.mean_squared_error(y_test, y_pred)
        num_params = len(regression.coef_) + 1  # features and intercept
        n = X_train.shape[0]
        aic = calculate_aic(n, mse, num_params)
        bic = calculate_bic(n, mse, num_params)
        if aic < min_aic:
            min_aic = aic
            current_bic = bic
            best_feature = feature
    selected_features = selected_features + [best_feature]
    print("Selected features: ", selected_features, "\nAIC: ", min_aic, "\nBIC: ", curr

y_pred = regression.predict(X_test[selected_features])
print("\n\nFinal r-squared: ", metrics.r2_score(y_test, y_pred))
```

```
Selected features:  ['weight']
AIC:  256.2464344506617
BIC:  263.16560547895006
Selected features:  ['weight', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year', 'year', 'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year', 'year', 'year', 'year',
'year']
AIC:  135.51868295158894
BIC:  145.89743949402143
Selected features:  ['weight', 'year', 'year', 'year', 'year', 'year', 'year', 'year',
'year', 'year']
AIC:  135.51868295158894
BIC:  145.89743949402143


Final r-squared:  0.8225451129806733
```

In [119…
```
regression.coef_
```

Out[119…
```
array([-9.99483913e+00, -1.58579584e+13, -1.71024346e+13, -4.99809099e+13,
        3.79471296e+13, -9.57780031e+12,  3.79471296e+13, -9.57780031e+12,
        3.65420012e+13, -1.03393569e+13])
```