DSC550 - Week4 - Predicting Fuel Efficiency - sidbhaumik

April 8, 2023

0.0.1 Build a linear regression model to predict fuel efficiency (miles per gallon) of automobiles

1. Load the data as a Pandas data frame and ensure that it imported correctly.

[34]:
```
# Import required Libraries

# to import the data
import pandas as pd

# to visualize
import seaborn as sns
import matplotlib.pyplot as plt

# to do math
import numpy as np
from scipy import stats
from scipy.stats import norm, skew

# to standardize the data
from sklearn.preprocessing import RobustScaler, StandardScaler

# to create the models
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
import xgboost as xgb

# to split the data
from sklearn.model_selection import train_test_split, GridSearchCV

# to calculate the error
from sklearn.metrics import mean_squared_error, accuracy_score,␣
↪mean_absolute_error,r2_score

# to average the models
from sklearn.base import clone

# to get rid of the warnings
import warnings
```

```
warnings.filterwarnings("ignore")
```

[3]:
```
# Loading the CSV data into a Pandas Dataframe
auto_df = pd.read_csv("auto-mpg.csv")
```

[4]:
```
# checking sample data
auto_df.head(3)
```

[4]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70

   origin                   car name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
```

2. Data Prep

[5]:
```
# Remove the car name column
auto_df = auto_df.drop(labels = ["car name"], axis = 1)
```

[35]:
```
auto_df.head(3)
```

[35]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70

   origin
0       1
1       1
2       1
```

[36]:
```
# Since we are predicting Fuel efficiency, so MPG is my Target column.
#auto_df = auto_df.rename(columns = {"mpg": "target"})
```

[6]:
```
# Checking column datatypes
auto_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   mpg            398 non-null    float64
 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   model year    398 non-null    int64
 7   origin        398 non-null    int64
dtypes: float64(3), int64(4), object(1)
memory usage: 25.0+ KB
```

[7]:
```
# Checking unique values in Horsepower column
auto_df['horsepower'].unique()
```

[7]:
```
array(['130', '165', '150', '140', '198', '220', '215', '225', '190',
       '170', '160', '95', '97', '85', '88', '46', '87', '90', '113',
       '200', '210', '193', '?', '100', '105', '175', '153', '180', '110',
       '72', '86', '70', '76', '65', '69', '60', '80', '54', '208', '155',
       '112', '92', '145', '137', '158', '167', '94', '107', '230', '49',
       '75', '91', '122', '67', '83', '78', '52', '61', '93', '148',
       '129', '96', '71', '98', '115', '53', '81', '79', '120', '152',
       '102', '108', '68', '58', '149', '89', '63', '48', '66', '139',
       '103', '125', '133', '138', '135', '142', '77', '62', '132', '84',
       '64', '74', '116', '82'], dtype=object)
```

[9]:
```
#dealing with missing values
auto_df = auto_df[auto_df['horsepower']!='?']
auto_df.horsepower = pd.to_numeric(auto_df.horsepower)
auto_df[['horsepower']] = auto_df[['horsepower']].replace(np.nan, auto_df.
 ↪horsepower.mean())
```

[10]:
```
# Datatype for Horsepower column changed from Object to Integer
auto_df.info()
```

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

Correlation Matrix

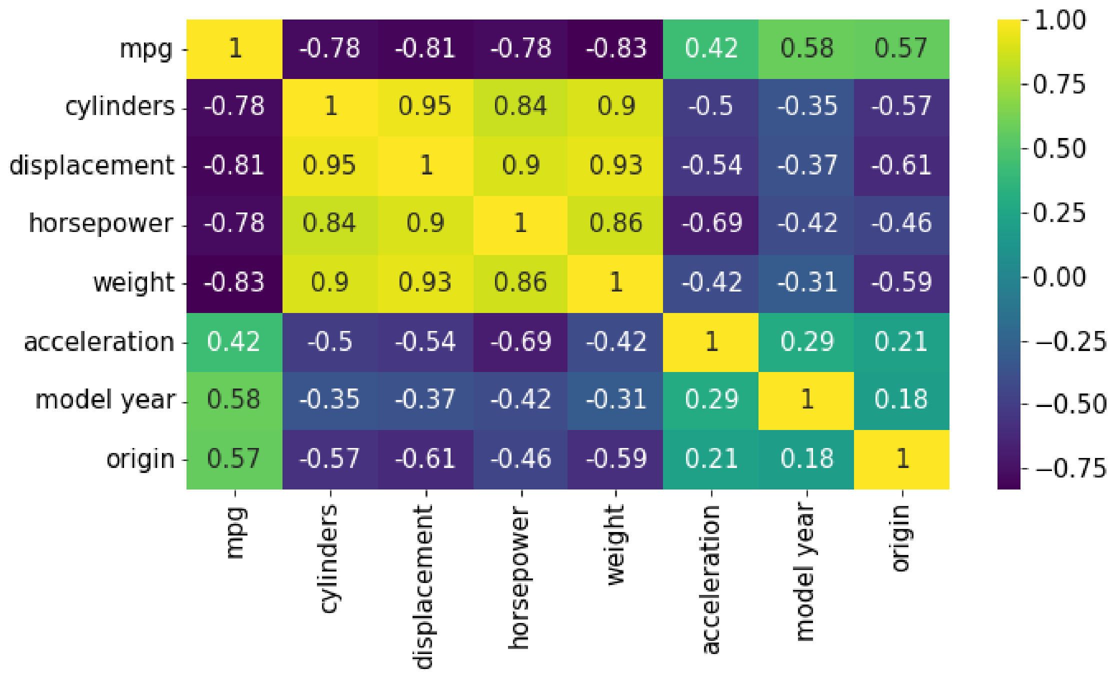
[15]:
```
print("The chart shows relation between target variable 'MPG' & other␣
 ↪predictive variables")
corr=auto_df.corr()
corr['mpg'].to_frame()
```

```
The chart shows relation between target variable 'MPG' & other predictive
variables
```

[15]:

	mpg
mpg	1.000000
cylinders	-0.777618
displacement	-0.805127
horsepower	-0.778427
weight	-0.832244
acceleration	0.423329
model year	0.580541
origin	0.565209

[17]:
```
# Correlation Heatmap
plt.rc('font',size=15)
plt.figure(figsize=(10,6))
sns.heatmap(corr,annot=True,cmap='viridis')
plt.tight_layout();
```

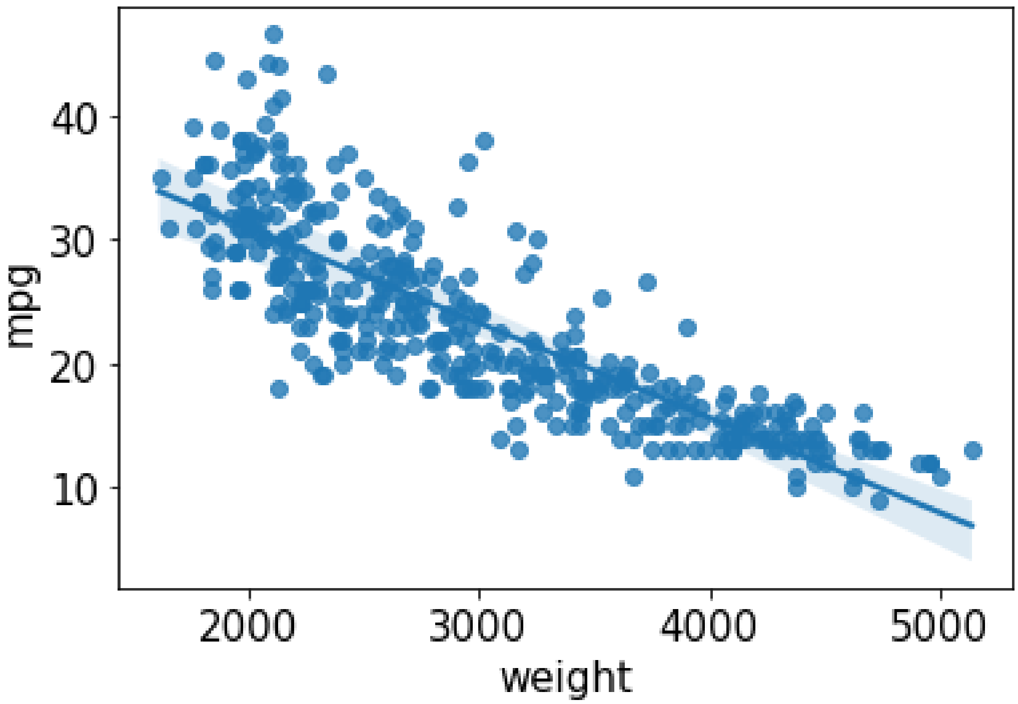


I see below 4 variables have strong negative correlation with MPG:

1. Weight
2. Displacement
3. Horepower
4. Cylinder

[19]:
```
# Corelation between MPG & Weight

corr_w_m = sns.regplot(data=auto_df,x='weight',y='mpg',ci=100,units='mpg')
```



A strong negative correlation between Weight & MPG. The higher the weight, the lower is the MPG and vice-versa

Randomly split the data into 80% training data and 20% test data, where your target is mpg.

[21]:
```
# split as train and test

x = auto_df.drop(["mpg"], axis = 1)
y = auto_df["mpg"]

testSize = 0.2
```

```
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = testSize,␣
↪random_state = 42)
```

[22]:
```
# standardization

scaler = StandardScaler()
xTrain = scaler.fit_transform(xTrain)
xTest = scaler.transform(xTest)
```

Train an ordinary linear regression on the training data.

Calculate R2, RMSE, and MAE on both the training and test sets and interpret your results.

[26]:
```
# create a dictionary to hold the rmse values

rmseValues = dict()
```

[30]:
```
# linear regression to fit the model

lr = LinearRegression()
lr.fit(xTrain, yTrain)

print("Linear Regression Coefficients: {}".format(lr.coef_))



# Calculating RMSE on Test data
yPredictedDummy = lr.predict(xTest)
rmse = mean_squared_error(yTest, yPredictedDummy, squared=False)

print("Linear Regression RMSE on Test Data:", rmse)

rmseValues["Linear Regression"] = rmse
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression RMSE on Test Data: 3.2727457003009564
```

[29]:
```
# Calculating RMSE on Training data

yPredictedDummy = lr.predict(xTrain)
rmse = mean_squared_error(yTrain, yPredictedDummy, squared=False)

print("Linear Regression RMSE on Training Data:", rmse)

rmseValues["Linear Regression"] = rmse
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression RMSE on Training Data: 3.3134960151437447
```

[32]:

```
# Calculating MAE on Training

lr = LinearRegression()
lr.fit(xTrain, yTrain)

print("Linear Regression Coefficients: {}".format(lr.coef_))

yPredictedDummy = lr.predict(xTrain)

mae = mean_absolute_error(yTrain, yPredictedDummy)

print("Linear Regression MAE on Training Data:", mae)
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression MAE on Training Data: 2.5481681962151366
```

[33]:

```
# Calculating MAE on Test data

lr = LinearRegression()
lr.fit(xTrain, yTrain)

print("Linear Regression Coefficients: {}".format(lr.coef_))

yPredictedDummy = lr.predict(xTest)

mae = mean_absolute_error(yTest, yPredictedDummy)

print("Linear Regression MAE on Test Data:", mae)
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression MAE on Test Data: 2.4197802491974536
```

[35]:

```
# Calculating R2 on Training

lr = LinearRegression()
lr.fit(xTrain, yTrain)

print("Linear Regression Coefficients: {}".format(lr.coef_))
```

```
yPredictedDummy = lr.predict(xTrain)

r2 = r2_score(yTrain, yPredictedDummy)

print("Linear Regression R2 on Training Data:", r2)
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression R2 on Training Data: 0.826001578671067
```

[36]:
```
# Calculating R2 on Test

lr = LinearRegression()
lr.fit(xTrain, yTrain)

print("Linear Regression Coefficients: {}".format(lr.coef_))

yPredictedDummy = lr.predict(xTest)

r2 = r2_score(yTest, yPredictedDummy)

print("Linear Regression R2 on Test Data:", r2)
```

```
Linear Regression Coefficients: [-0.58705525  1.56527255 -0.81420529 -5.15767051
0.10676699  2.78255456
  1.30024012]
Linear Regression R2 on Test Data: 0.7901500386760345
```

XGboost Regression Model

[40]:
```
# xgboost

xgbModel = xgb.XGBRegressor()

tunedParams = {"nthread": [4],
               "objective": ["reg:squarederror"],
               "learning_rate": [0.03, 0.05, 0.07],
               "max_depth": [5, 6, 7],
               "min_child_weight": [4],
               "subsample": [0.7],
               "colsample_bytree": [0.7],
               "n_estimators": [500, 1000]}
nFolds = 5

clf = GridSearchCV(xgbModel,
                   tunedParams,
                   cv = nFolds,
```

```
                   scoring = "neg_mean_squared_error",
                   refit = True,
                   n_jobs = 5)
clf.fit(xTrain, yTrain)

xgbModel = clf.best_estimator_

yPredictedDummy = clf.predict(xTest)
rmse = mean_squared_error(yTest, yPredictedDummy, squared=False)

print("XGBoost RMSE on Test Data:", rmse)
```

```
XGBoost RMSE on Test Data: 2.434097937120446
```

[42]:
```
yPredictedDummy = clf.predict(xTrain)
rmse = mean_squared_error(yTrain, yPredictedDummy, squared=False)

print("XGBoost RMSE on Training Data:", rmse)
```

```
XGBoost RMSE on Training Data: 0.20223194351569815
```

[44]:
```
yPredictedDummy = clf.predict(xTest)
mae = mean_absolute_error(yTest, yPredictedDummy)

print("XGBoost MAE on Test Data:", mae)
```

```
XGBoost MAE on Test Data: 1.7401606137239483
```

[45]:
```
yPredictedDummy = clf.predict(xTrain)
mae = mean_absolute_error(yTrain, yPredictedDummy)

print("XGBoost MAE on Training Data:", mae)
```

```
XGBoost MAE on Training Data: 0.15520055362591728
```

[46]:
```
yPredictedDummy = clf.predict(xTest)
r2 = r2_score(yTest, yPredictedDummy)

print("XGBoost R2 on Test Data:", r2)
```

```
XGBoost R2 on Test Data: 0.8839191797701402
```

[47]:
```
yPredictedDummy = clf.predict(xTrain)
r2 = r2_score(yTrain, yPredictedDummy)

print("XGBoost R2 on Train Data:", r2)
```

```
XGBoost R2 on Train Data: 0.9993518553898141
```

We can see with XGBoost we are getting lesser values for RMSE & MAE. But for R2 score, XGBpoost is giving slightly higer value