Open in Colab

Fuel Efficiency Prediction

Purpose

The automotive industry is extremely competitive. With increasing fuel prices and picky consumers, automobile makers are constantly optimizing their processes to increase fuel efficiency.But, what if you could have a reliable estimator for a car’s mpg given some known specifications about the vehicle? Then, you could beat a competitor to market by both having a more desirable vehicle that is also more efficient, reducing wasted R&D costs, and gaining large chunks of the market.

Utilizing machine learning, Cocolevio can help you build prediction models designed to give you an edge over your competitors

Introduction

Having a decent comprehension of what influences fuel utilization, and afterward having the option to foresee it, is vital to upgrading eco-friendliness. In the transportation business, the Miles per Gallon, or MPG, is utilized to work out a vehicle's proficiency as a component of the energy it consumes.

Dataset

Dataset link: https://www.kaggle.com/datasets/uciml/autompg-dataset

Importing required Liabraries

In [68]:
```
from __future__ import absolute_import, division, print_function
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import mean_squared_error
```

Loading dataset using pandas liabrary

In [69]:
```
df = pd.read_csv('/content/auto-mpg.csv')
```

Checking dataset shape & information

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

Out[70]: (398, 9)

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 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  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

In [72]:

```
df.head()
```

Out[72]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

Data Cleaning

We can see that dataset contains charater values in 'car name' column. So we have toremove this column from dataset usind pandas drop method.

In [73]:

```
df = df.drop(['car name'],axis = 1)
```

In [74]:

```
df.head()
```

Out[74]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	18.0	8	307.0	130	3504	12.0	70	1
1	15.0	8	350.0	165	3693	11.5	70	1
2	18.0	8	318.0	150	3436	11.0	70	1
3	16.0	8	304.0	150	3433	12.0	70	1
4	17.0	8	302.0	140	3449	10.5	70	1

Checking null value in dataset

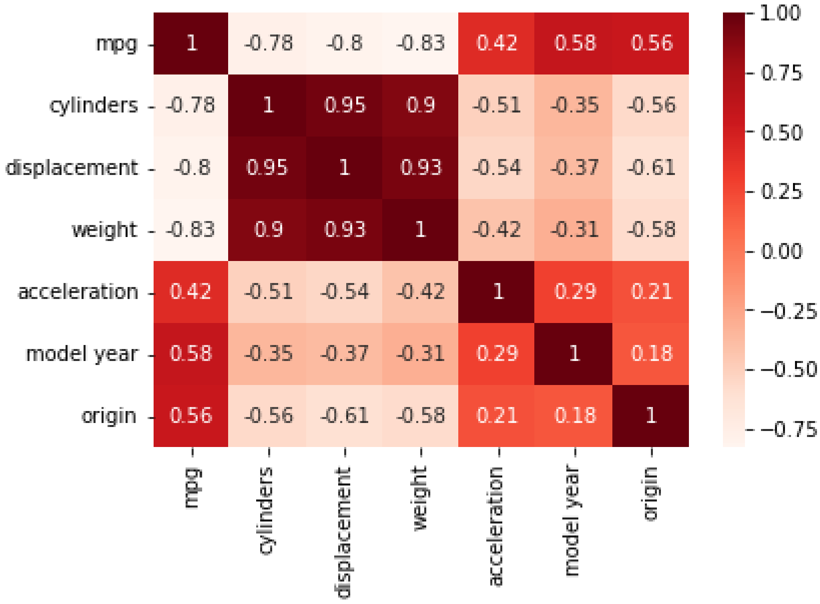
In [75]:
```
df.isna().sum()
```

Out[75]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
dtype: int64
```

EDA

corelation matrix for features

In [76]:
```
plt.figure(figsize=(6,4))
cor = df.corr()
sns.heatmap(cor, annot = True, cmap = plt.cm.Reds)
plt.show()
```



Relating with output variable

In [77]:
```
#Correlation with output variable
cor_target = abs(cor['mpg'])
#Selecting highly correlated features
relavent_features = cor_target[cor_target>0.5]
relavent_features
```

Out[77]:
```
mpg             1.000000
cylinders       0.775396
displacement    0.804203
weight          0.831741
model year      0.579267
origin          0.563450
Name: mpg, dtype: float64
```

The "Origin" column is really categorical, not numeric. So convert that to a one-hot

In [78]:

```
origin = df.pop('origin')
```

In [79]:

```
df['USA'] = (origin==1)*1.0
df['Europe'] = (origin==2)*1.0
df['Japan'] = (origin==3)*1.0
```

In [80]:

```
df['horsepower'] = df['horsepower'].replace(to_replace = '?', value =105)
df['horsepower'] = df['horsepower'].astype(float)
```

In [81]:

```
df.head()
```

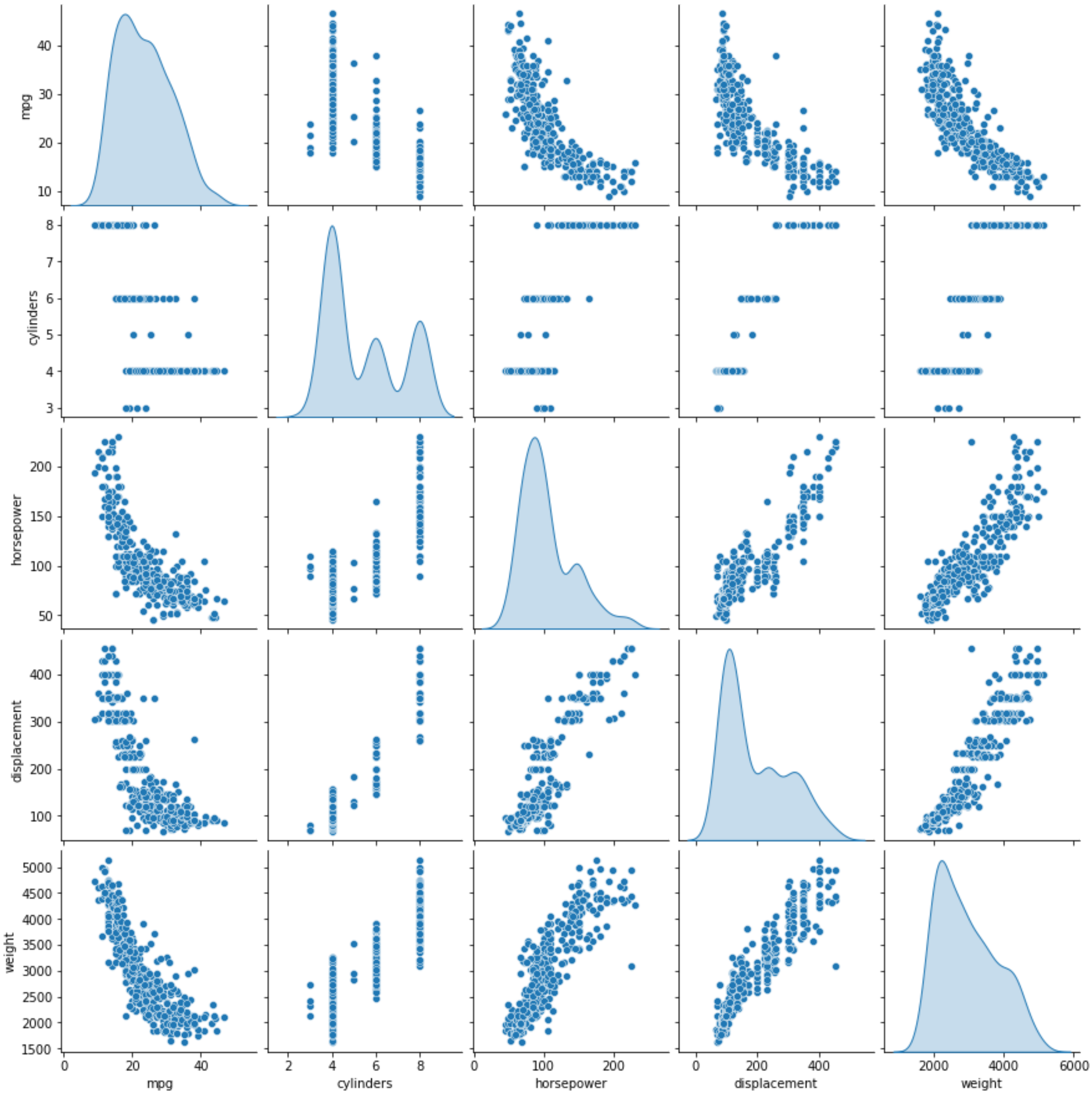
Out[81]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	USA	Europe	Japar
0	18.0	8	307.0	130.0	3504	12.0	70	1.0	0.0	0.(
1	15.0	8	350.0	165.0	3693	11.5	70	1.0	0.0	0.(
2	18.0	8	318.0	150.0	3436	11.0	70	1.0	0.0	0.(
3	16.0	8	304.0	150.0	3433	12.0	70	1.0	0.0	0.(
4	17.0	8	302.0	140.0	3449	10.5	70	1.0	0.0	0.(

In [43]:

```
sns.pairplot(df[["mpg", "cylinders","horsepower", "displacement", "weight"]], d
```

Out[43]: <seaborn.axisgrid.PairGrid at 0x7fe0749429a0>



In [44]:

```
df_stats = df.describe()
df_stats.pop("mpg")
df_stats = df_stats.transpose()
df_stats
```

Out[44]:

	count	mean	std	min	25%	50%	75%	max
cylinders	398.0	5.454774	1.701004	3.0	4.000	4.0	8.000	8.0
displacement	398.0	193.425879	104.269838	68.0	104.250	148.5	262.000	455.0
horsepower	398.0	104.477387	38.199242	46.0	76.000	95.0	125.000	230.0
weight	398.0	2970.424623	846.841774	1613.0	2223.750	2803.5	3608.000	5140.0
acceleration	398.0	15.568090	2.757689	8.0	13.825	15.5	17.175	24.8
model year	398.0	76.010050	3.697627	70.0	73.000	76.0	79.000	82.0
USA	398.0	0.625628	0.484569	0.0	0.000	1.0	1.000	1.0
Europe	398.0	0.175879	0.381197	0.0	0.000	0.0	0.000	1.0
Japan	398.0	0.198492	0.399367	0.0	0.000	0.0	0.000	1.0

In [45]:

```
X = df.drop(['mpg'], axis = 1)
```

```
Y = df['mpg']
```

Splitting dataset into train and test

In [46]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, random_state=10, test_siz
```

In [47]:
```
X_train.shape
```

Out[47]: (318, 9)

In [48]:
```
X_train.head()
```

Out[48]:

	cylinders	displacement	horsepower	weight	acceleration	model year	USA	Europe	Japan
303	4	85.0	65.0	2020	19.2	79	0.0	0.0	1.0
347	4	85.0	65.0	1975	19.4	81	0.0	0.0	1.0
149	4	120.0	97.0	2489	15.0	74	0.0	0.0	1.0
100	6	250.0	88.0	3021	16.5	73	1.0	0.0	0.0
175	4	90.0	70.0	1937	14.0	75	0.0	1.0	0.0

In [49]:
```
Y_train.shape
```

Out[49]: (318,)

Loading models to make prediction

lightgbm model

In [50]:
```
import lightgbm as lgb
```

Building model

In [51]:
```
params = {
    'task': 'train',
    'boosting': 'gbdt',
    'objective': 'regression',
    'num_leaves': 10,
    'learnnig_rage': 0.05,
    'metric': {'l2','l1'},
    'verbose': -1
}
```

Loading Data

In [52]:
```
lgb_train = lgb.Dataset(X_train, Y_train)
lgb_test = lgb.Dataset(X_test, Y_test)
```

Fit the model using training dataset

In [53]:

```
model1 = lgb.train(params,
                   train_set=lgb_train,
                   valid_sets=lgb_test,
                   early_stopping_rounds=50)
```

```
[1]	valid_0's l2: 51.2169	valid_0's l1: 6.02569
Training until validation scores don't improve for 50 rounds.
[2]	valid_0's l2: 42.8875	valid_0's l1: 5.47208
[3]	valid_0's l2: 36.3846	valid_0's l1: 4.9916
[4]	valid_0's l2: 30.8799	valid_0's l1: 4.55334
[5]	valid_0's l2: 26.0557	valid_0's l1: 4.14178
[6]	valid_0's l2: 22.51	valid_0's l1: 3.83761
[7]	valid_0's l2: 19.631	valid_0's l1: 3.55343
[8]	valid_0's l2: 17.0105	valid_0's l1: 3.2666
[9]	valid_0's l2: 15.2441	valid_0's l1: 3.06476
[10]	valid_0's l2: 13.7179	valid_0's l1: 2.88228
[11]	valid_0's l2: 12.6141	valid_0's l1: 2.75135
[12]	valid_0's l2: 11.7395	valid_0's l1: 2.64718
[13]	valid_0's l2: 10.9539	valid_0's l1: 2.54565
[14]	valid_0's l2: 10.3067	valid_0's l1: 2.47033
[15]	valid_0's l2: 9.81338	valid_0's l1: 2.40809
[16]	valid_0's l2: 9.47483	valid_0's l1: 2.36082
[17]	valid_0's l2: 9.15844	valid_0's l1: 2.3265
[18]	valid_0's l2: 8.97987	valid_0's l1: 2.28878
[19]	valid_0's l2: 8.81615	valid_0's l1: 2.25614
[20]	valid_0's l2: 8.6848	valid_0's l1: 2.22431
[21]	valid_0's l2: 8.64037	valid_0's l1: 2.20288
[22]	valid_0's l2: 8.52279	valid_0's l1: 2.17722
[23]	valid_0's l2: 8.45621	valid_0's l1: 2.15867
[24]	valid_0's l2: 8.45798	valid_0's l1: 2.14677
[25]	valid_0's l2: 8.3907	valid_0's l1: 2.12966
[26]	valid_0's l2: 8.34034	valid_0's l1: 2.12046
[27]	valid_0's l2: 8.36848	valid_0's l1: 2.11936
[28]	valid_0's l2: 8.37117	valid_0's l1: 2.11785
[29]	valid_0's l2: 8.4405	valid_0's l1: 2.12432
[30]	valid_0's l2: 8.42782	valid_0's l1: 2.12967
[31]	valid_0's l2: 8.41697	valid_0's l1: 2.12671
[32]	valid_0's l2: 8.39526	valid_0's l1: 2.1244
[33]	valid_0's l2: 8.45727	valid_0's l1: 2.13712
[34]	valid_0's l2: 8.46272	valid_0's l1: 2.14496
[35]	valid_0's l2: 8.53309	valid_0's l1: 2.16007
[36]	valid_0's l2: 8.48266	valid_0's l1: 2.15465
[37]	valid_0's l2: 8.48117	valid_0's l1: 2.16193
[38]	valid_0's l2: 8.4492	valid_0's l1: 2.15621
[39]	valid_0's l2: 8.4918	valid_0's l1: 2.16705
[40]	valid_0's l2: 8.51376	valid_0's l1: 2.16651
[41]	valid_0's l2: 8.55759	valid_0's l1: 2.16729
[42]	valid_0's l2: 8.59404	valid_0's l1: 2.16898
[43]	valid_0's l2: 8.59404	valid_0's l1: 2.16876
[44]	valid_0's l2: 8.63315	valid_0's l1: 2.17645
[45]	valid_0's l2: 8.61844	valid_0's l1: 2.17499
[46]	valid_0's l2: 8.65743	valid_0's l1: 2.17704
[47]	valid_0's l2: 8.67704	valid_0's l1: 2.18124
[48]	valid_0's l2: 8.6547	valid_0's l1: 2.17757
[49]	valid_0's l2: 8.69421	valid_0's l1: 2.18172
[50]	valid_0's l2: 8.71998	valid_0's l1: 2.18442
[51]	valid_0's l2: 8.74161	valid_0's l1: 2.18943
[52]	valid_0's l2: 8.75293	valid_0's l1: 2.18742
[53]	valid_0's l2: 8.69851	valid_0's l1: 2.18446
[54]	valid_0's l2: 8.68149	valid_0's l1: 2.18536
[55]	valid_0's l2: 8.68528	valid_0's l1: 2.1861
[56]	valid_0's l2: 8.67969	valid_0's l1: 2.18697
[57]	valid_0's l2: 8.68458	valid_0's l1: 2.18994
```

```
[58]	valid_0's l2: 8.71028	valid_0's l1: 2.19531
[59]	valid_0's l2: 8.75022	valid_0's l1: 2.20056
[60]	valid_0's l2: 8.76207	valid_0's l1: 2.19968
[61]	valid_0's l2: 8.79361	valid_0's l1: 2.20688
[62]	valid_0's l2: 8.80226	valid_0's l1: 2.20617
[63]	valid_0's l2: 8.73345	valid_0's l1: 2.19885
[64]	valid_0's l2: 8.74069	valid_0's l1: 2.20164
[65]	valid_0's l2: 8.74203	valid_0's l1: 2.20473
[66]	valid_0's l2: 8.78281	valid_0's l1: 2.21407
[67]	valid_0's l2: 8.7945	valid_0's l1: 2.21491
[68]	valid_0's l2: 8.78422	valid_0's l1: 2.20813
[69]	valid_0's l2: 8.76369	valid_0's l1: 2.20435
[70]	valid_0's l2: 8.79072	valid_0's l1: 2.21003
[71]	valid_0's l2: 8.80996	valid_0's l1: 2.2146
[72]	valid_0's l2: 8.80287	valid_0's l1: 2.21262
[73]	valid_0's l2: 8.80999	valid_0's l1: 2.21107
[74]	valid_0's l2: 8.83093	valid_0's l1: 2.2151
[75]	valid_0's l2: 8.79597	valid_0's l1: 2.21289
[76]	valid_0's l2: 8.77739	valid_0's l1: 2.21191
Early stopping, best iteration is:
[26]	valid_0's l2: 8.34034	valid_0's l1: 2.12046
```

Predicting test dataset

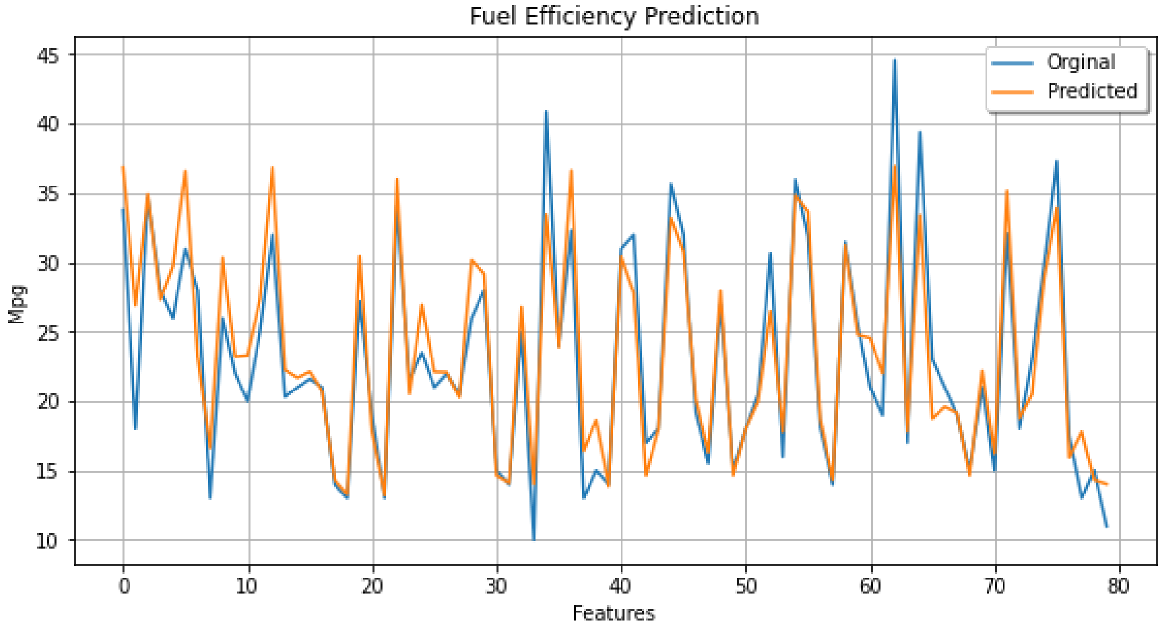
In [54]:

```
Y_pred1 = model1.predict(X_test)
# accuracy check
mse1 = mean_squared_error(Y_test, Y_pred1)
rmse1 = mse1**(0.5)
print("MSE1: %.2f" % mse1)
print("RMSE1: %.2f" % rmse1)
```

```
MSE1: 8.34
RMSE1: 2.89
```

In [82]:

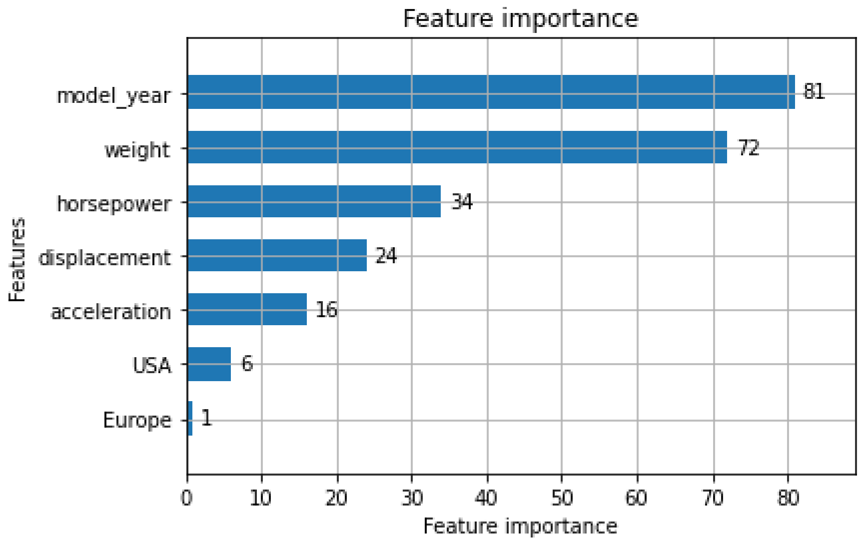
```
x_as = range(len(Y_test))
plt.figure(figsize=(10,5))
plt.plot(x_as, Y_test, label='Orginal')
plt.plot(x_as, Y_pred1, label='Predicted')
plt.title('Fuel Efficiency Prediction')
plt.xlabel('Features')
plt.ylabel('Mpg')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



Plot feature importance of lightgbm

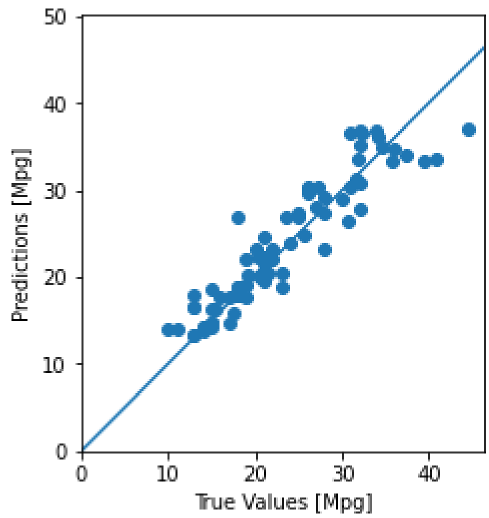
In [55]:
```
lgb.plot_importance(model1, height=0.6)
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe090fc7e50>



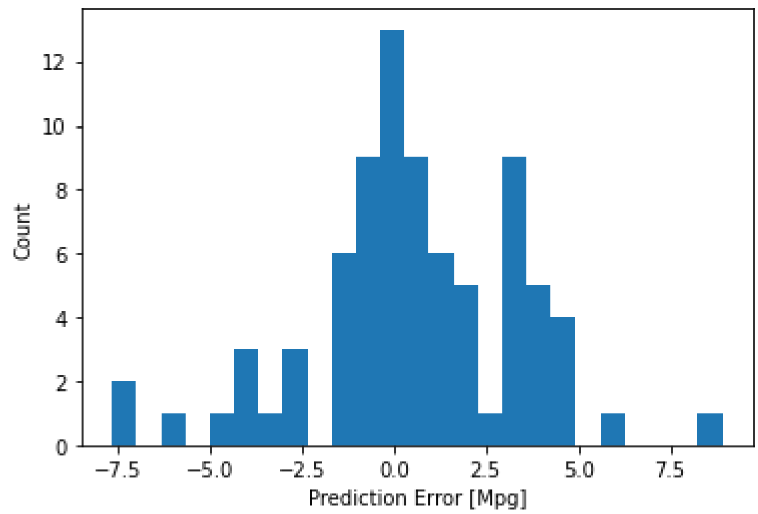
Visualizing in plot

In [56]:
```
plt.scatter(Y_test, Y_pred1)
plt.xlabel('True Values [Mpg]')
plt.ylabel('Predictions [Mpg]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```



In [57]:

```
error = Y_pred1 - Y_test
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [Mpg]")
_ = plt.ylabel("Count")
```



XGBregressor model

In [58]:

```
import xgboost as xgb
from xgboost import plot_importance
```

Training model

In [59]:

```
model2 = xgb.XGBRegressor(n_estimators=100, max_depth=6, learning_rate=0.1)
model2.fit(X_train, Y_train,
            eval_set=[(X_train, Y_train), (X_test, Y_test)],
            early_stopping_rounds=50)
```

```
[17:33:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:22.0706       validation_1-rmse:21.5671
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for earl
y stopping.

Will train until validation_1-rmse hasn't improved in 50 rounds.
[1]     validation_0-rmse:19.9603       validation_1-rmse:19.4536
[2]     validation_0-rmse:18.0566       validation_1-rmse:17.5614
```

```
[3]	validation_0-rmse:16.3384	validation_1-rmse:15.8315
[4]	validation_0-rmse:14.7926	validation_1-rmse:14.2802
[5]	validation_0-rmse:13.402	validation_1-rmse:12.8893
[6]	validation_0-rmse:12.1546	validation_1-rmse:11.6899
[7]	validation_0-rmse:11.0274	validation_1-rmse:10.5709
[8]	validation_0-rmse:10.0123	validation_1-rmse:9.58037
[9]	validation_0-rmse:9.09865	validation_1-rmse:8.67439
[10]	validation_0-rmse:8.27837	validation_1-rmse:7.92317
[11]	validation_0-rmse:7.54034	validation_1-rmse:7.18336
[12]	validation_0-rmse:6.88015	validation_1-rmse:6.59136
[13]	validation_0-rmse:6.285 validation_1-rmse:6.00407
[14]	validation_0-rmse:5.7527	validation_1-rmse:5.54299
[15]	validation_0-rmse:5.26965	validation_1-rmse:5.12071
[16]	validation_0-rmse:4.83836	validation_1-rmse:4.78264
[17]	validation_0-rmse:4.44725	validation_1-rmse:4.46414
[18]	validation_0-rmse:4.09563	validation_1-rmse:4.21112
[19]	validation_0-rmse:3.77962	validation_1-rmse:3.99148
[20]	validation_0-rmse:3.49072	validation_1-rmse:3.81154
[21]	validation_0-rmse:3.23031	validation_1-rmse:3.64446
[22]	validation_0-rmse:2.99851	validation_1-rmse:3.50426
[23]	validation_0-rmse:2.77853	validation_1-rmse:3.40322
[24]	validation_0-rmse:2.58142	validation_1-rmse:3.33079
[25]	validation_0-rmse:2.40917	validation_1-rmse:3.25084
[26]	validation_0-rmse:2.24785	validation_1-rmse:3.20552
[27]	validation_0-rmse:2.10046	validation_1-rmse:3.16878
[28]	validation_0-rmse:1.96528	validation_1-rmse:3.12512
[29]	validation_0-rmse:1.84593	validation_1-rmse:3.08058
[30]	validation_0-rmse:1.73507	validation_1-rmse:3.05219
[31]	validation_0-rmse:1.6367	validation_1-rmse:3.02894
[32]	validation_0-rmse:1.5447	validation_1-rmse:3.01868
[33]	validation_0-rmse:1.46044	validation_1-rmse:3.02398
[34]	validation_0-rmse:1.38998	validation_1-rmse:3.03542
[35]	validation_0-rmse:1.32641	validation_1-rmse:3.02912
[36]	validation_0-rmse:1.26713	validation_1-rmse:3.02931
[37]	validation_0-rmse:1.21629	validation_1-rmse:3.02863
[38]	validation_0-rmse:1.16438	validation_1-rmse:3.03408
[39]	validation_0-rmse:1.12141	validation_1-rmse:3.04009
[40]	validation_0-rmse:1.07923	validation_1-rmse:3.04273
[41]	validation_0-rmse:1.04443	validation_1-rmse:3.04986
[42]	validation_0-rmse:1.01046	validation_1-rmse:3.05075
[43]	validation_0-rmse:0.975881	validation_1-rmse:3.05824
[44]	validation_0-rmse:0.950507	validation_1-rmse:3.06622
[45]	validation_0-rmse:0.924352	validation_1-rmse:3.06658
[46]	validation_0-rmse:0.900951	validation_1-rmse:3.07304
[47]	validation_0-rmse:0.87535	validation_1-rmse:3.06549
[48]	validation_0-rmse:0.857326	validation_1-rmse:3.06882
[49]	validation_0-rmse:0.839809	validation_1-rmse:3.07142
[50]	validation_0-rmse:0.825392	validation_1-rmse:3.07488
[51]	validation_0-rmse:0.813913	validation_1-rmse:3.079
[52]	validation_0-rmse:0.795791	validation_1-rmse:3.07659
[53]	validation_0-rmse:0.782728	validation_1-rmse:3.07911
[54]	validation_0-rmse:0.771689	validation_1-rmse:3.08125
[55]	validation_0-rmse:0.7569	validation_1-rmse:3.0796
[56]	validation_0-rmse:0.749628	validation_1-rmse:3.08095
[57]	validation_0-rmse:0.743282	validation_1-rmse:3.08022
[58]	validation_0-rmse:0.732893	validation_1-rmse:3.07967
[59]	validation_0-rmse:0.720177	validation_1-rmse:3.0767
[60]	validation_0-rmse:0.714853	validation_1-rmse:3.0761
[61]	validation_0-rmse:0.70829	validation_1-rmse:3.07421
[62]	validation_0-rmse:0.69935	validation_1-rmse:3.074
[63]	validation_0-rmse:0.689904	validation_1-rmse:3.07424
[64]	validation_0-rmse:0.685563	validation_1-rmse:3.07322
[65]	validation_0-rmse:0.676983	validation_1-rmse:3.07243
[66]	validation_0-rmse:0.672616	validation_1-rmse:3.07167
```

```
[67]	validation_0-rmse:0.660425	validation_1-rmse:3.07417
[68]	validation_0-rmse:0.6576	validation_1-rmse:3.07497
[69]	validation_0-rmse:0.653832	validation_1-rmse:3.07527
[70]	validation_0-rmse:0.644787	validation_1-rmse:3.08213
[71]	validation_0-rmse:0.641982	validation_1-rmse:3.08373
[72]	validation_0-rmse:0.638424	validation_1-rmse:3.08352
[73]	validation_0-rmse:0.632505	validation_1-rmse:3.0797
[74]	validation_0-rmse:0.624917	validation_1-rmse:3.08528
[75]	validation_0-rmse:0.620513	validation_1-rmse:3.08619
[76]	validation_0-rmse:0.616895	validation_1-rmse:3.09111
[77]	validation_0-rmse:0.613557	validation_1-rmse:3.09021
[78]	validation_0-rmse:0.607806	validation_1-rmse:3.08981
[79]	validation_0-rmse:0.599988	validation_1-rmse:3.08895
[80]	validation_0-rmse:0.595802	validation_1-rmse:3.08944
[81]	validation_0-rmse:0.587563	validation_1-rmse:3.0875
[82]	validation_0-rmse:0.582584	validation_1-rmse:3.08774
Stopping. Best iteration:
[32]	validation_0-rmse:1.5447	validation_1-rmse:3.01868
```

Out[59]:

```
XGBRegressor(max_depth=6)
```

After training the model, we'll check the model training score

In [60]:

```python
score = model2.score(X_train, Y_train)
print("Training score of XGBregressor: ", score)
```

```
Training score of XGBregressor:  0.960763226476405
```

We can also apply the cross-validation method to evaluate the training score

In [61]:

```python
scores = cross_val_score(model2, X_train, Y_train,cv=10)
print("Mean cross-validation score of XGBregressor: %.2f" % scores.mean())
```

```
[17:33:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Mean cross-validation score of XGBregressor: 0.87
```

Or if you want to use the KFlold method in cross-validation it goes as below

In [62]:

```python
kfold = KFold(n_splits=10, shuffle=True)
kf_cv_scores = cross_val_score(model2, X_train, Y_train, cv=kfold )
print("K-fold CV average score of XGBregressor: %.2f" % kf_cv_scores.mean())
```

```
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
[17:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
K-fold CV average score of XGBregressor: 0.85
```

Test dataset prediction and MSE and RMSE calculation

In [63]:

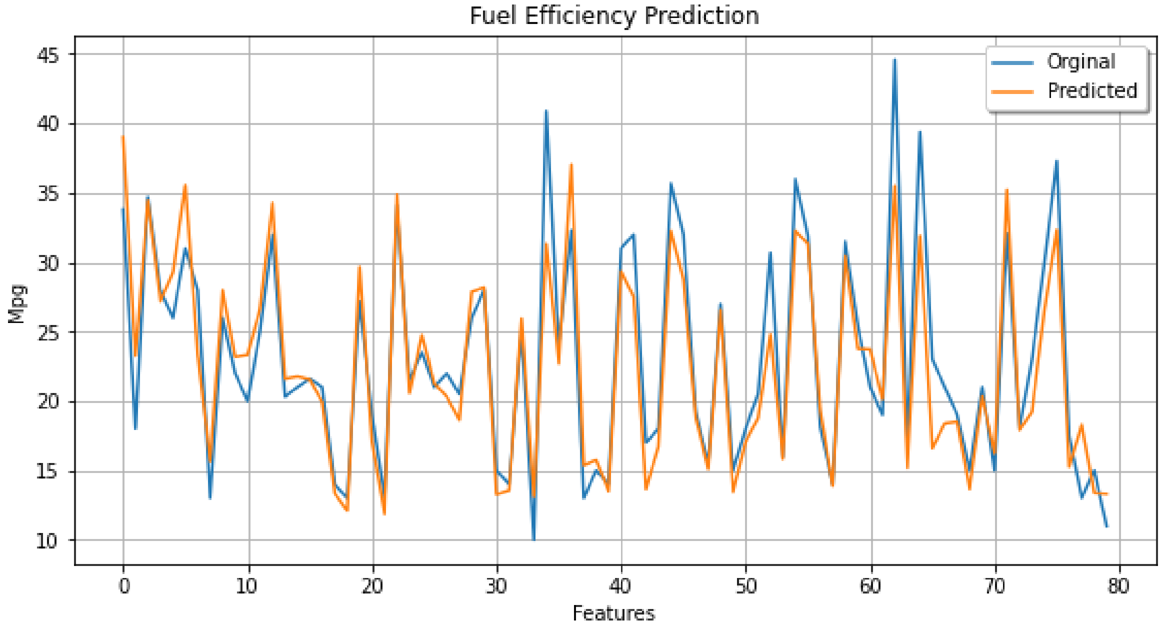
```
Y_pred2 = model2.predict(X_test)
# accuracy check
mse2 = mean_squared_error(Y_test, Y_pred2)
print("MSE: %.2f" % mse2)
rmse2 = mse2**(0.5)
print("RMSE2: %.2f" % rmse2)
```

```
MSE: 9.11
RMSE2: 3.02
```

Finally, we'll visualize the original and predicted test data in a plot to compare visually

In [64]:

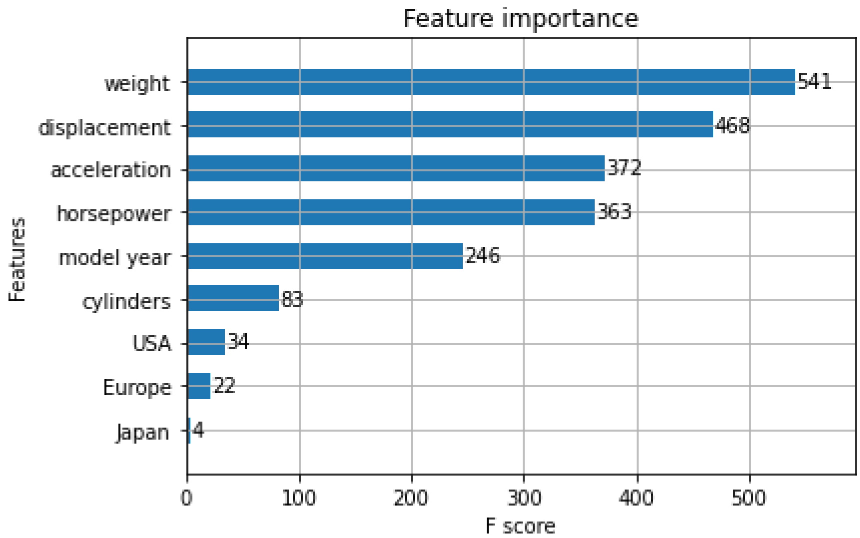
```
x_as = range(len(Y_test))
plt.figure(figsize=(10,5))
plt.plot(x_as, Y_test, label='Orginal')
plt.plot(x_as, Y_pred2, label='Predicted')
plt.title('Fuel Efficiency Prediction')
plt.xlabel('Features')
plt.ylabel('Mpg')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



Plot feature importance of XGBregressor

In [65]:
```
plot_importance(model2, height=0.6)
```

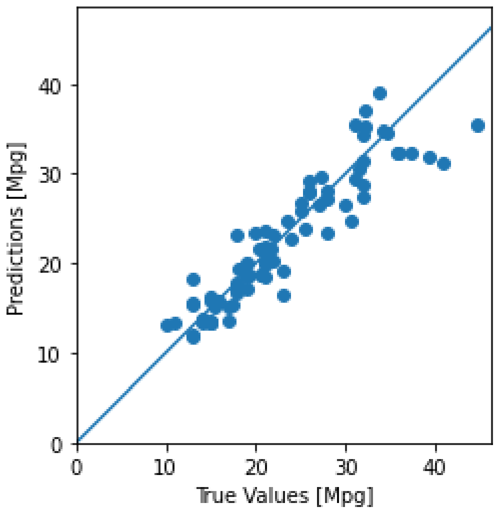
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe06b4aa3a0>



Lightgbm gives less RSME score than XGBregressor.

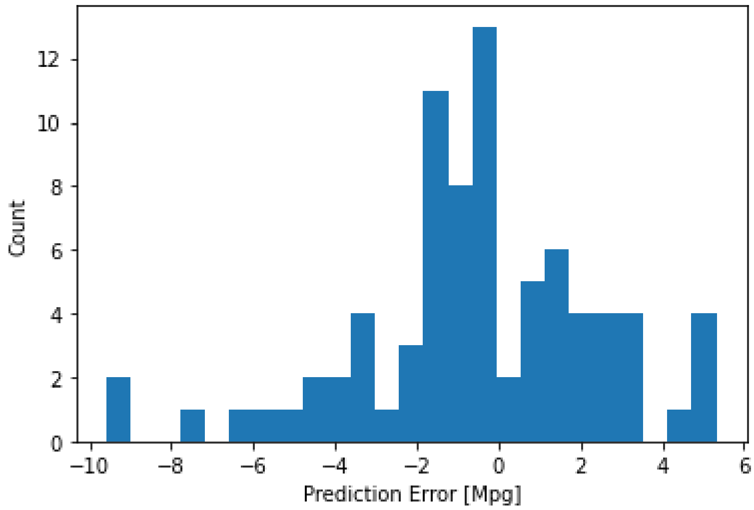
Visualizing in plot

In [66]:
```
plt.scatter(Y_test, Y_pred2)
plt.xlabel('True Values [Mpg]')
plt.ylabel('Predictions [Mpg]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```



In [67]:

```
error = Y_pred2 - Y_test
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [Mpg]")
_ = plt.ylabel("Count")
```



Conclusion

From the above model training using "*lightgbm*" and "*XGBregressor*" models I found that lightgbm model has less RSME(Root Mean Squared Error) = 2.89 than XGBregressor model RSME = 3.02.