Car Price Prediction Model

This dataset is from the "Car Features and MSRP" dataset by CooperUnion on Kaggle. The original dataset features 16 columns of variables for almost 12,000 rows of observations. The data was originally scraped from Edmunds and Twitter. This dataset can be found at: https://www.kaggle.com/CooperUnion/cardataset (https://www.kaggle.com/CooperUnion/cardataset). The data has been previously cleaned and manipulated as part of a previous project where the dataset was explored and feature correlations were investigated. The current version of the dataset contains 25 columns and 9197 rows.

Import and Prepare Dataset

In [1]:
```
# Import some packages and set plots to be embedded inline
import numpy as np
import pandas as pd
from pandas.api.types import CategoricalDtype
import matplotlib.pyplot as plt
import seaborn as sb

%matplotlib inline
```

In [2]:
```
# Import csv file of dataset into pandas df and preview

cars = pd.read_csv('cars_final.csv')
cars = cars.drop(columns = 'Unnamed: 0')
```

In [3]:
```
# Define function to inspect data frames. Prints first few lines, determines s
ize/shape of data frame,
# shows descriptive statistics, shows data types, shows missing or incomplete
 data, check for duplicate data.

def inspect_df(df):
    print('Header:')
    print('{}'.format(df.head()))
    print()
    print('Shape: {}'.format(df.shape))
    print()
    print('Statistics:')
    print('{}'.format(df.describe()))
    print()
    print('Info:')
    print('{}'.format(df.info()))

# Use inspect_df on cars df
inspect_df(cars)
```

```
Header:
  make        model  year              fuel   hp  cylinders   trans  \
0  BMW  1 Series M  2011  premium unleaded   335          6  manual
1  BMW    1 Series  2011  premium unleaded   300          6  manual
2  BMW    1 Series  2011  premium unleaded   300          6  manual
3  BMW    1 Series  2011  premium unleaded   230          6  manual
4  BMW    1 Series  2011  premium unleaded   230          6  manual

              drive  doors    vsize  ... comb_mpg    log_hp  log_hwy_mpg  \
0  rear wheel drive      2  Compact  ...     22.5  2.525045     1.414973
1  rear wheel drive      2  Compact  ...     23.5  2.477121     1.447158
2  rear wheel drive      2  Compact  ...     24.0  2.477121     1.447158
3  rear wheel drive      2  Compact  ...     23.0  2.361728     1.447158
4  rear wheel drive      2  Compact  ...     23.0  2.361728     1.447158

   log_city_mpg  log_comb_mpg  log_popularity  log_price         ppp  \
0      1.278754      1.352183        3.592843   4.664031  137.716418
1      1.278754      1.371068        3.592843   4.609061  135.500000
2      1.301030      1.380211        3.592843   4.560504  121.166667
3      1.255273      1.361728        3.592843   4.469085  128.043478
4      1.255273      1.361728        3.592843   4.537819  150.000000

   mpg_ratio  mpg_per_hp
0   1.368421    0.067164
1   1.473684    0.078333
2   1.400000    0.080000
3   1.555556    0.100000
4   1.555556    0.100000

[5 rows x 25 columns]

Shape: (9197, 25)

Statistics:
              year           hp    cylinders        doors      hwy_mpg  \
count  9197.000000  9197.000000  9197.000000  9197.000000  9197.000000
mean   2012.839948   239.385561     5.342068     3.592476    27.448625
std       4.491115    80.144919     1.411283     0.789131     7.631527
min    2001.000000    74.000000     0.000000     2.000000    13.000000
25%    2010.000000   175.000000     4.000000     4.000000    23.000000
50%    2015.000000   237.000000     6.000000     4.000000    27.000000
75%    2016.000000   292.000000     6.000000     4.000000    31.000000
max    2017.000000   707.000000    10.000000     4.000000   354.000000

          city_mpg   popularity         price     comb_mpg       log_hp  \
count  9197.000000  9197.000000   9197.000000  9197.000000  9197.000000
mean     20.273024  1574.347505  33420.121779    23.860824     2.354574
std       6.790177  1436.753110  12579.714860     6.888741     0.147841
min      10.000000    21.000000   9299.000000    11.500000     1.869232
25%      16.000000   549.000000  24165.000000    19.500000     2.243038
50%      19.000000  1385.000000  31040.000000    23.000000     2.374748
75%      23.000000  2009.000000  40445.000000    27.000000     2.465383
max     137.000000  5657.000000  75000.000000   189.000000     2.849419

       log_hwy_mpg  log_city_mpg  log_comb_mpg  log_popularity    log_price
\
count  9197.000000   9197.000000   9197.000000     9197.000000  9197.000000
```

```
mean      1.426093      1.290547      1.364050         3.010463     4.494422
std       0.101634      0.112987      0.105296         0.446193     0.160639
min       1.113943      1.000000      1.060698         1.322219     3.968436
25%       1.361728      1.204120      1.290035         2.739572     4.383187
50%       1.431364      1.278754      1.361728         3.141450     4.491922
75%       1.491362      1.361728      1.431364         3.302980     4.606865
max       2.549003      2.136721      2.276462         3.752586     4.875061

               ppp    mpg_ratio   mpg_per_hp
count  9197.000000  9197.000000  9197.000000
mean    141.277584     1.372781     0.118153
std      31.577367     0.181985     0.072449
min      72.392857     0.810219     0.024045
25%     119.580052     1.312500     0.069697
50%     136.877049     1.375000     0.094714
75%     157.692308     1.440000     0.151351
max     311.711712    14.750000     0.846429

Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9197 entries, 0 to 9196
Data columns (total 25 columns):
make              9197 non-null object
model             9197 non-null object
year              9197 non-null int64
fuel              9197 non-null object
hp                9197 non-null int64
cylinders         9197 non-null int64
trans             9197 non-null object
drive             9197 non-null object
doors             9197 non-null int64
vsize             9197 non-null object
body              9197 non-null object
hwy_mpg           9197 non-null int64
city_mpg          9197 non-null int64
popularity        9197 non-null int64
price             9197 non-null int64
comb_mpg          9197 non-null float64
log_hp            9197 non-null float64
log_hwy_mpg       9197 non-null float64
log_city_mpg      9197 non-null float64
log_comb_mpg      9197 non-null float64
log_popularity    9197 non-null float64
log_price         9197 non-null float64
ppp               9197 non-null float64
mpg_ratio         9197 non-null float64
mpg_per_hp        9197 non-null float64
dtypes: float64(10), int64(8), object(7)
memory usage: 1.8+ MB
None
```

In [4]:
```
# Drop unwanted columns (manufactured features for previous analysis)
cars = cars.drop(columns = ['log_hp', 'log_hwy_mpg', 'log_city_mpg', 'log_comb
_mpg', 'log_popularity', 'log_price', 'ppp'])
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9197 entries, 0 to 9196
Data columns (total 18 columns):
make           9197 non-null object
model          9197 non-null object
year           9197 non-null int64
fuel           9197 non-null object
hp             9197 non-null int64
cylinders      9197 non-null int64
trans          9197 non-null object
drive          9197 non-null object
doors          9197 non-null int64
vsize          9197 non-null object
body           9197 non-null object
hwy_mpg        9197 non-null int64
city_mpg       9197 non-null int64
popularity     9197 non-null int64
price          9197 non-null int64
comb_mpg       9197 non-null float64
mpg_ratio      9197 non-null float64
mpg_per_hp     9197 non-null float64
dtypes: float64(3), int64(8), object(7)
memory usage: 1.3+ MB
```

In [5]:
```
# Make list of columns with type object
objects = list(cars.select_dtypes(include=['object']).columns)
objects
```

Out[5]:
```
['make', 'model', 'fuel', 'trans', 'drive', 'vsize', 'body']
```

In [6]:
```
# Convert vsize column to ordinal encoding
vsize_mapper = {'Compact':1, 'Midsize':2, 'Large':3}

cars['vsize'] = cars['vsize'].replace(vsize_mapper)

cars.vsize.value_counts()
```

Out[6]:
```
2    3674
1    3538
3    1985
Name: vsize, dtype: int64
```

In [7]:
```
# Use One Hot encoding to replace other object columns
cars = pd.get_dummies(cars, columns=['make', 'model', 'fuel', 'trans', 'drive'
, 'body'],
               prefix=['make', 'model', 'fuel', 'trans', 'drive', 'body'])

cars.head()
```

Out[7]:

	year	hp	cylinders	doors	vsize	hwy_mpg	city_mpg	popularity	price	comb_mpg	...	b
0	2011	335	6	2	1	26	19	3916	46135	22.5	...	
1	2011	300	6	2	1	28	19	3916	40650	23.5	...	
2	2011	300	6	2	1	28	20	3916	36350	24.0	...	
3	2011	230	6	2	1	28	18	3916	29450	23.0	...	
4	2011	230	6	2	1	28	18	3916	34500	23.0	...	

5 rows × 688 columns

In [8]:
```
# Rearrange columns so price is first column
cols = list(cars)
cols.insert(0, cols.pop(cols.index('price')))
cars = cars.loc[:, cols]
```

In [9]:
```
# Split labels and features, then split train and test sets (20% reserved for
 testing)
from sklearn import model_selection
array = cars.values
X = array[:,1:]
Y = array[:,0]
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test
_size=0.2, random_state=0)
```

Build Regression Model

In [10]:
```
# Import ML models
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.linear_model import SGDRegressor
from sklearn.svm import LinearSVR
from sklearn import metrics
```

In [11]:
```
# Make list of Regression models to try, then use loop to test them
models = []
models.append(('Linear', LinearRegression()))
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('ENet', ElasticNet()))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))
models.append(('SVR', LinearSVR(max_iter=1000000)))

results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=0)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=k
fold, scoring='r2')
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Linear: -3012376878991.356445 (6021668537198.416992)
Ridge: 0.920128 (0.019164)
Lasso: 0.920815 (0.016796)
ENet: 0.703104 (0.011496)
SGD: -6786216467077053965926 4.000000 (957641151155488308592 64.000000)
SVR: 0.663518 (0.016567)
```

In [12]:
```
# Scale data to range 0-1 with MinMaxScaler then split train and test
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_scaled,
Y, test_size=0.2, random_state=0)
```

In [13]:
```
# Make list of Regression models to try, then use loop to test them
models = []
models.append(('Linear', LinearRegression()))
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('ENet', ElasticNet()))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))
models.append(('SVR', LinearSVR(max_iter=10000)))

results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=0)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=k
fold, scoring='r2')
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Linear: -2956743032063867748352.000000 (5517588585807564767232.000000)
Ridge: 0.929964 (0.002424)
Lasso: 0.929754 (0.002052)
ENet: 0.419442 (0.011645)
SGD: 0.920971 (0.004640)
SVR: -0.947042 (0.042046)
```

In [14]:
```
from sklearn.feature_selection import SelectPercentile, f_regression
selector = SelectPercentile(f_regression, percentile=50)
X_50 = selector.fit_transform(X,Y)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_50, Y, t
est_size=0.2, random_state=0)
```

In [15]:
```
# Make list of Regression models to try, then use loop to test them
models = []
models.append(('Linear', LinearRegression()))
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('ENet', ElasticNet()))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))
models.append(('SVR', LinearSVR(max_iter=1000000)))

results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=0)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=k
fold, scoring='r2')
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Linear: -31461405271257.617188 (62922810542517.015625)
Ridge: 0.899819 (0.022660)
Lasso: 0.902783 (0.021105)
ENet: 0.702267 (0.011629)
SGD: -122591848044680718581 76.000000 (100220215877287395983 36.000000)
SVR: 0.662964 (0.016300)
```

In [16]:
```
from sklearn.feature_selection import SelectPercentile, f_regression
selector = SelectPercentile(f_regression, percentile=50)
X_50 = selector.fit_transform(X_scaled,Y)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_50, Y, t
est_size=0.2, random_state=0)
```

In [17]:
```
# Make list of Regression models to try, then use loop to test them
models = []
models.append(('Linear', LinearRegression()))
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('ENet', ElasticNet()))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))
models.append(('SVR', LinearSVR(max_iter=10000)))

results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=0)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=k
fold, scoring='r2')
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Linear: -801132840258313912320.000000 (1596720913636653006848.000000)
Ridge: 0.911785 (0.003746)
Lasso: 0.912215 (0.002978)
ENet: 0.418579 (0.011642)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:1229: ConvergenceWarning: Maximum number of iteration reached befor
e convergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)

SGD: 0.901389 (0.004881)
SVR: -1.163010 (0.037427)
```

In [18]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=100)
X_pca = pca.fit_transform(X_scaled)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_pca, Y,
test_size=0.2, random_state=0)
```

In [19]:
```
# Make list of Regression models to try, then use loop to test them
models = []
models.append(('Linear', LinearRegression()))
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('ENet', ElasticNet()))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))
models.append(('SVR', LinearSVR(max_iter=10000)))

results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=0)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=k
fold, scoring='r2')
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Linear: 0.838785 (0.003232)
Ridge: 0.838804 (0.003186)
Lasso: 0.838778 (0.003194)
ENet: 0.418981 (0.011653)
SGD: 0.838683 (0.003305)
SVR: -4.800288 (0.081802)
```

Obtained best results so far with scaled features and no dimensionality reduction. Best results from Ridge, Lasso, and SGD.

In [20]:
```
# Scale data to range 0-1 with MinMaxScaler then split train and test
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_scaled,
Y, test_size=0.2, random_state=0)
```

In [21]:
```
# Test 3 best models on test data
from sklearn.metrics import r2_score

models = []
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso(max_iter=10000)))
models.append(('SGD', SGDRegressor(max_iter=1000, tol=0.001)))

for name, model in models:
    clf = model
    clf.fit(X_train, Y_train)
    pred = clf.predict(X_test)
    r2 = r2_score(Y_test, pred)
    msg = "%s: %s" % (name, r2)
    print(msg)
```

```
Ridge: 0.9354618869524685
Lasso: 0.9333913113477414
SGD: 0.9233876853697575
```

Ridge and Lasso give best performance, use GridSearchCV to tune hyperparameters for these two models.

In [22]:
```
# Use GridSearchCV to tune hyperparameters, make classifier with best params a
nd test on test data
from sklearn.model_selection import GridSearchCV
params = {'alpha' : (np.arange(0.1, 1, 0.1)), 'solver' : ('auto', 'svd', 'chol
esky', 'lsqr', 'sparse_cg', 'sag', 'saga')}
alg = Ridge(random_state=0)
clf = GridSearchCV(alg, params, cv = 5, scoring = 'r2', n_jobs = -1)
clf.fit(X_train, Y_train)
print("Best Parameters:", clf.best_params_)
pred = clf.predict(X_test)
print("R2:", r2_score(Y_test, pred))
```

```
Best Parameters: {'alpha': 0.1, 'solver': 'lsqr'}
R2: 0.9380702637464594
```

In [23]:
```
# Use GridSearchCV to tune hyperparameters, make classifier with best params a
nd test on test data
params = {'alpha' : (np.arange(0.1, 2, 0.1))}
alg = Lasso(random_state=0, max_iter=10000)
clf = GridSearchCV(alg, params, cv = 5, scoring = 'r2', n_jobs = -1)
clf.fit(X_train, Y_train)
print("Best Parameters:", clf.best_params_)
pred = clf.predict(X_test)
print("R2:", r2_score(Y_test, pred))
```

```
Best Parameters: {'alpha': 0.6}
R2: 0.9361679033984465
```

After tuning hyperparameters, obtained best results with min/max scaled features, no dimensionality reduction, and Ridge Regression. The optimal parameters for the Ridge Regression are alpha = 0.1, solver = lsqr, and the rest defaults.

Final Model

In [24]:
```
# Make array of data, split into features and labels, split into train and tes
t sets (20% reserved for testing)
array = cars.values
X = array[:,1:]
Y = array[:,0]
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test
_size=0.2, random_state=0)
```

In [25]:
```
# Use pipeline to combine scaler and regression model
from sklearn.pipeline import Pipeline
scaler = MinMaxScaler()
reg = Ridge(alpha = 0.1, solver = 'lsqr')
scale_ridge = Pipeline([('scaler', scaler), ('ridge', reg)])
scale_ridge = scale_ridge.fit(X_train, Y_train)
```

In [26]:
```
# Test on testing set and evaluate performance
print('R2 Score:', scale_ridge.score(X_test, Y_test))
```

```
R2 Score: 0.9379168297253947
```

The final model consists of a MinMaxScaler coupled with a Ridge Regression with alpha = 0.1, solver = lsqr, and defaults for other hyperparameters. This model gives an R2 score of 0.9379 when testing on the reserved test data set.

In [28]:
```
# Save dataset and regression model as pickles
import pickle
cars.to_pickle('cars_ML_dataset.pkl')
pickle.dump(scale_ridge, open('cars_ML_model.pkl', 'wb'))
```