Predicting used car prices with linear regression

Dr. Bruns

In this notebook we'll use experiment with linear regression in the prediction of used car prices. We'll try different models, use polynomial features, and implement forward feature selection.

Instructions:

- problems for you to insert code are indicated with lines that begin with #@ followed by a problem number
- always use a 75/25 split when splitting the data into training/test sets
- always use 'random_state = 0' when using test_train_split so that you get the same answers as the model output

Out[2]: Click here to display/hide the code.

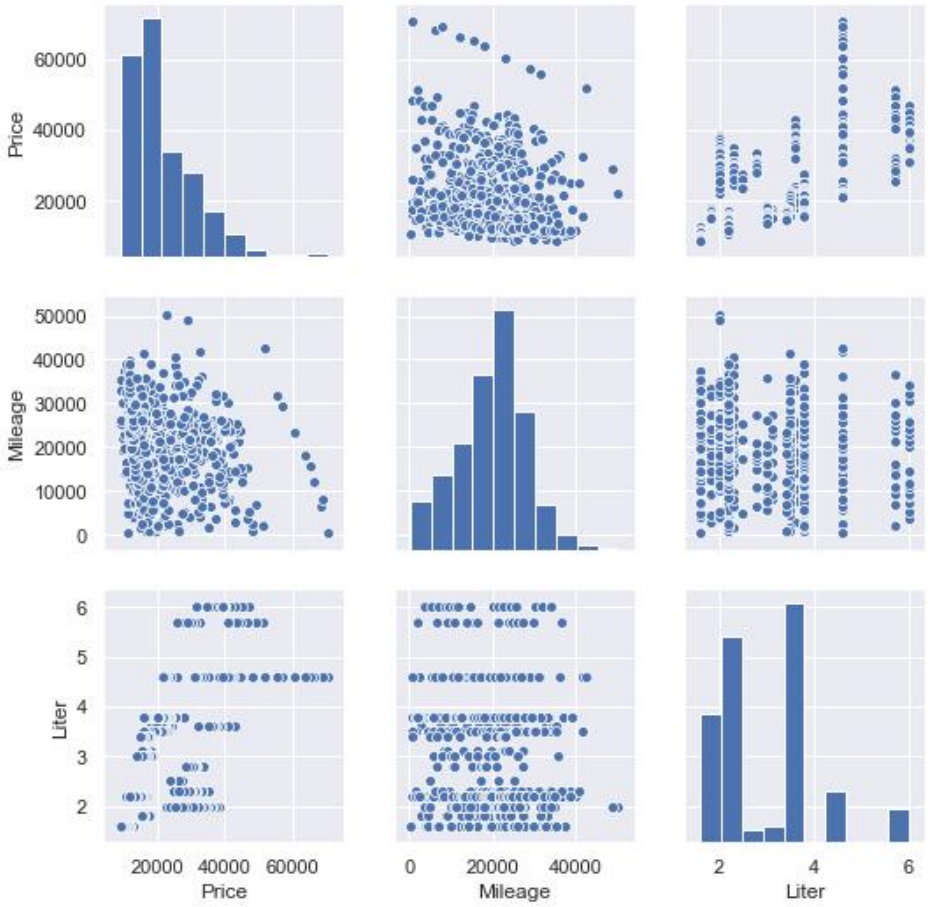
Read the Kuiper's 2008 car data

We'll drop the Model and Trim features, as we are interested in making predictions without identifying the exact kind of car.

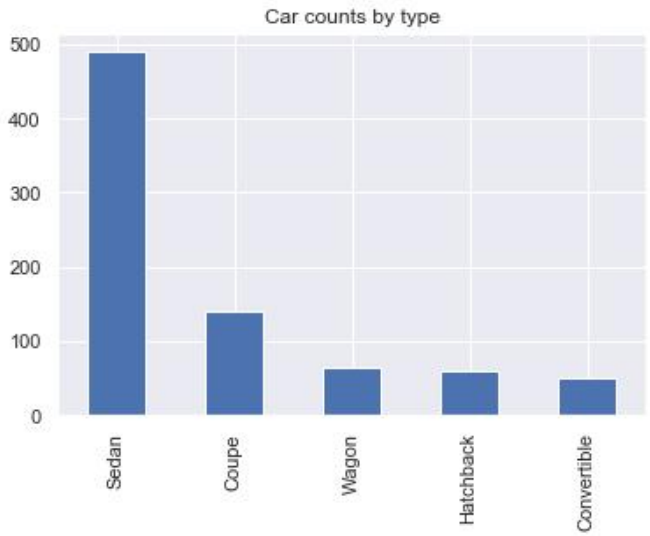
Are there any NA values in the data set?

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 10 columns):
Price       804 non-null float64
Mileage     804 non-null int64
Make        804 non-null object
Type        804 non-null object
Cylinder    804 non-null int64
Liter       804 non-null float64
Doors       804 non-null int64
Cruise      804 non-null int64
Sound       804 non-null int64
Leather     804 non-null int64
dtypes: float64(2), int64(6), object(2)
memory usage: 62.9+ KB
```

There appears to be no missing data; at least none in the form of NA values. Let's look at the relationships between some of the features.



What kinds of cars are in the data set and what are their proportions?

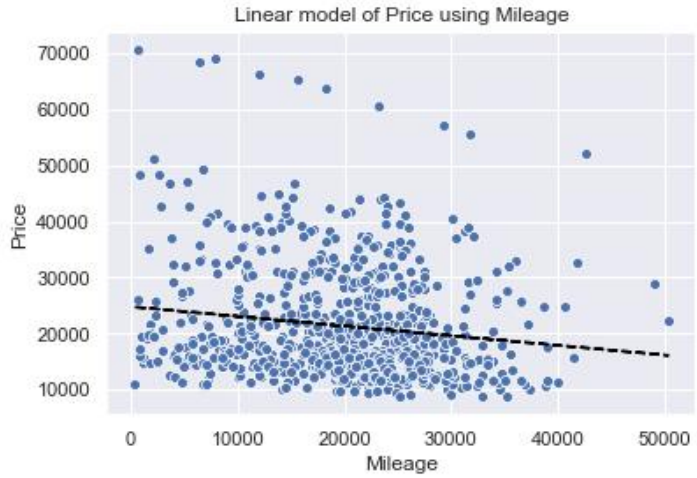


Let's build a model to predict a used car's price from its mileage.

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

With one predictor, our linear model can be shown as a line. Let's look at the model compared to the data.

Out[9]: `Text(0.5, 1.0, 'Linear model of Price using Mileage')`



What are the coefficients of the model?

```
intercept: 24764.56
coefficient for Mileage: -0.17
r-squared value: 0.02
```

The model says that for every extra mile of milage, our prediction for price will go down by 17 cents! It makes sense that more milage means a lower price. Let's build another model to predict price, but this time using cruise control and leather interior as additional features.

Out[11]: `LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)`

Here are the coefficients of our new model.

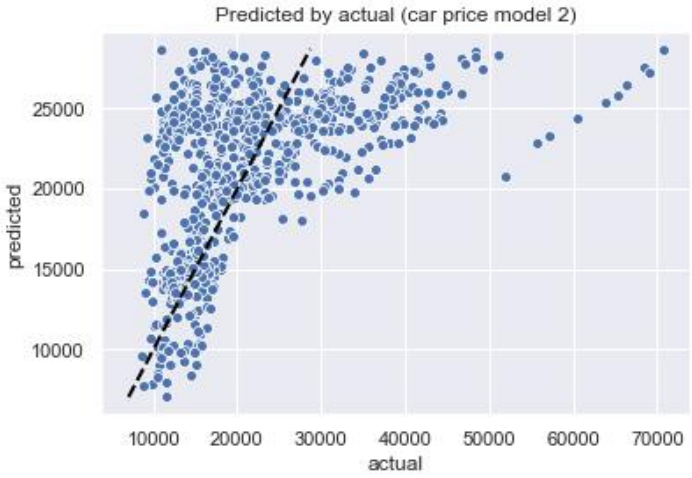
```
intercept: 24764.56
coefficients:
  Mileage: -0.19
  Cruise: 10256.12
  Leather: 4175.58
```

Amazingly, the coefficient for cruise control indicates that -- in this model -- the presence of cruise control will increase the predicted price by over $10,000. That seems crazy, but maybe cruise control is a good differentiator between a cheap car and an expensive car.

What would this model predict for the price of a car with 20,000 miles and cruise control but not a leather interior?

```
20827.73
```

A good way to see how well a regression model is working is to plot predicted values against actual values. In our case we'll plot predicted price vs. actual price. Ideally, all points will be close to the line showing where actual=predicted.



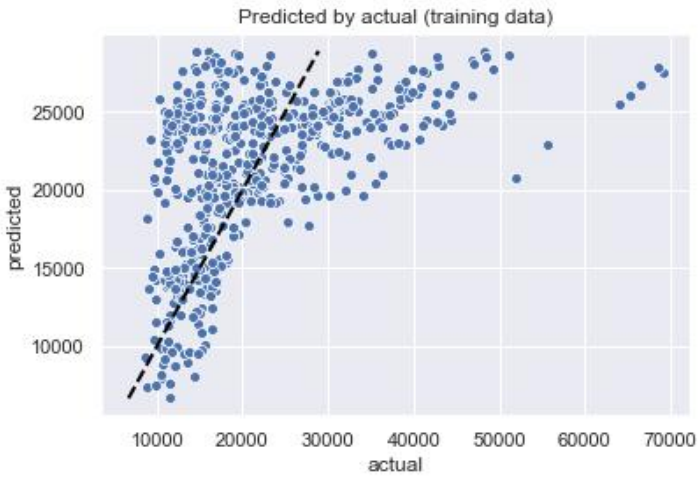
The plot shows that many predictions are bad -- they are far off the line. The points on the far right are cases in which our predicted price is much less than the actual price. For example, in one of these points the actual price is about 60,000 dollars but our predicted price is about 24,000 dollars. In the other direction, there's a car with an actual price of about 10,000 in which our model predicts about 25,000. You wouldn't want to use this model in deciding how much to pay for a used car.

We've been using one data set both to train our model and to make predictions. It's much better to split the data into separate training and set sets.

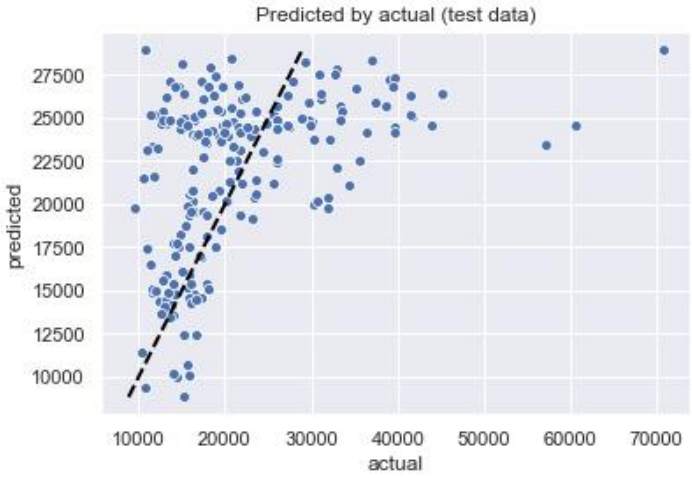
Now let's fit our same model using only training data.

Out[17]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Let's plot predicted vs. actual for predictions made on the training set. This should be similar to our last predicted vs. actual plot.



Now let's do a predicted vs. actual plot for predictions made on the test data.



The two plots look pretty similar. Again we see some really bad predictions, like the predicted price of about 28,000 for a car having an actual price of 70,000.

A good way to measure the goodness of a regression model is by using the root mean squared error.

```
RMSE: 8517.23
```

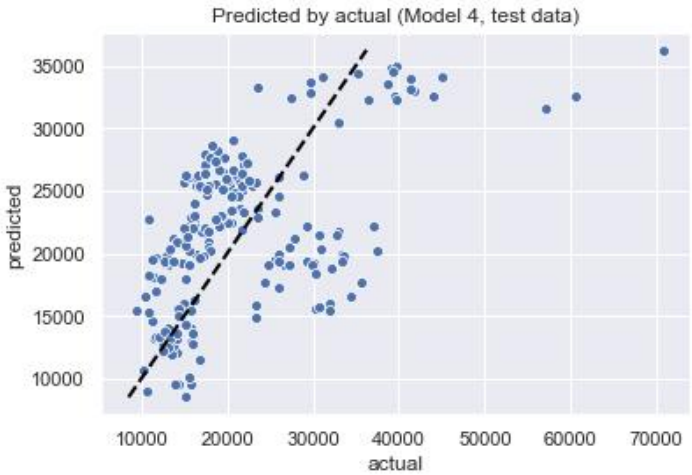
We can make better predictions if we use more features as predictors. Let's use a feature that represents the number of cylinders in a car's engine.

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Let's look at the RMSE of the new model, as well as the R-squared statistic.

```
RMSE of reg4: 7810.77
r-squared value of reg4: 0.4489
```

The new cylinder feature reduces the RMSE by over 10 percent. Let's look at predicted vs. actual for the new model.



Some of the predictions are still really bad. Would scaling the data help us make better predictions?

Let's check the R-squared score of our new model with scaled data. How does it compare to the R-squared value above for the unscaled data?

```
r-squared value of regs: 0.4489
```

With Scikit-Learn, all features must be numeric. Let's now transform all categorical variables into numeric variables using the dummy variables approach.

A check to ensure that we now have only numeric variables.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 17 columns):
Price             804 non-null float64
Mileage           804 non-null int64
Cylinder          804 non-null int64
Liter             804 non-null float64
Doors             804 non-null int64
Cruise            804 non-null int64
Sound             804 non-null int64
Leather           804 non-null int64
Make_Cadillac     804 non-null uint8
Make_Chevrolet    804 non-null uint8
Make_Pontiac      804 non-null uint8
Make_SAAB         804 non-null uint8
Make_Saturn       804 non-null uint8
Type_Coupe        804 non-null uint8
Type_Hatchback    804 non-null uint8
Type_Sedan        804 non-null uint8
Type_Wagon        804 non-null uint8
dtypes: float64(2), int64(6), uint8(9)
memory usage: 57.4 KB
```

Let's build a linear model in which we use all of the features as predictors, and look at the values of all the coefficients.

```
intercept: 34393.88
coefficients:
  Mileage: -0.19
  Cylinder: -1217.73
  Liter: 5674.20
  Doors: -5338.01
  Cruise: 149.11
  Sound: 150.82
  Leather: 112.58
  Make_Cadillac: 15638.25
  Make_Chevrolet: -1616.20
  Make_Pontiac: -1831.79
  Make_SAAB: 10623.51
  Make_Saturn: -1345.82
  Type_Coupe: -12563.40
  Type_Hatchback: -2209.53
  Type_Sedan: -2221.69
  Type_Wagon: 1762.21
```

It is interesting to look at the coefficient values. We see that Saab command a premium price, and that coupes are associated with a low price compared to sedans. None of the features like cruise control, sound system, or leather seats make a big different in the predictions from this model.

```
R-squared: 0.94
RMSE: 2685.22
```

We can add even more features to the model by adding derived features. Let's try the PolynomialFeatures class.

For a sanity check, how many rows and columns in the new data set?

Out[31]: (804, 153)

Let's go for broke and build a model using *all* features. What is the RMSE on the test data for such a model?

```
RMSE: 1446.13
```

This RMSE value is much lower than before, but our model has 153 features! It would be good to know which feature is the most important. As an experiment, let's look at the RMSE if we use only the first feature.

```
RMSE for feature 0 only: 9983.69
```

Now we'll compute the RMSE for each individual feature, and see which is lowest.

Which single feature has the lowest RMSE?

```
best feature: x2 x7, best RMSE: 7206.27
```

Which is the best set of 10 features? The number of possible sets of 10 features is equal to 153 * 152 * 151 * ... * 144, which is a gigantic number (bigger than a billion billion). Instead we can use forward feature selection, which is a kind of greedy method.

```
num features: 1; rmse: 7206.27
num features: 2; rmse: 5896.86
num features: 3; rmse: 4003.42
num features: 4; rmse: 3568.84
num features: 5; rmse: 3322.06
num features: 6; rmse: 2462.93
num features: 7; rmse: 2293.72
num features: 8; rmse: 2167.47
num features: 9; rmse: 2067.19
num features: 10; rmse: 2009.69
```

How does the RMSE of the model created using the 10 features found with forward feature selection compare the the RMSE of the model that uses all the features?

```
test RMSE with 10 features: 2075.86
```