[1]
```python
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
from scipy.stats import zscore
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, cross_val_score
```

[2]
```python
# code in this cell from:
# https://stackoverflow.com/questions/27934885/how-to-hide-code-from-cells-in-ipython-notebook-visualized-with-nbviewer
from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here to display/hide the code."></form>''')
```

Click here to display/hide the code.

```python
# switch to seaborn default stylistic parameters
sns.set()
sns.set_context('notebook')
```

```python
df = pd.read_csv("https://raw.githubusercontent.com/grbruns/cst383/master/kuiper-2008-cars.csv")
df.drop(['Model', 'Trim'], inplace=True, axis=1)
```

Are there any NA values in the data set?

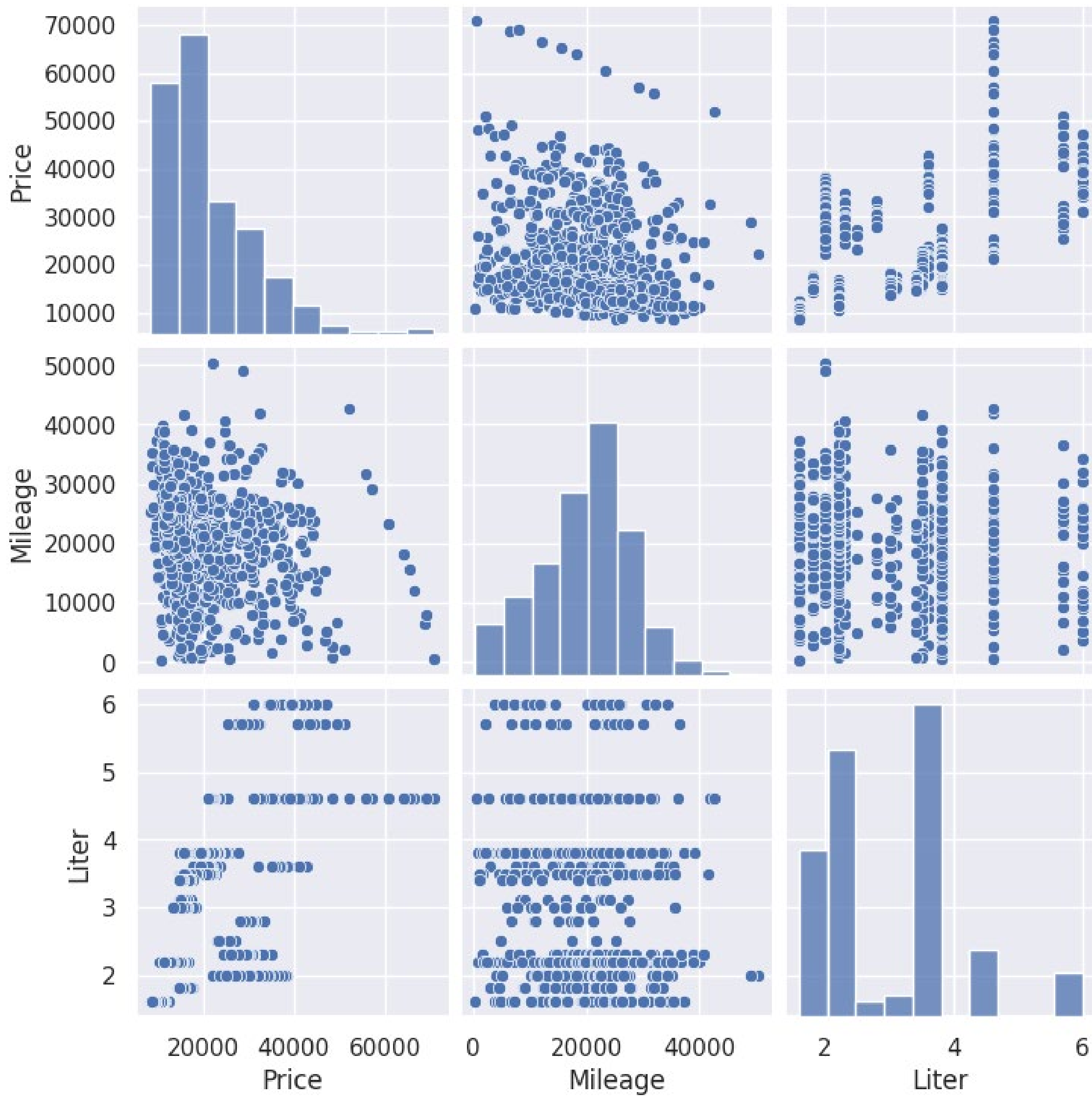
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Price     804 non-null    float64
 1   Mileage   804 non-null    int64  
 2   Make      804 non-null    object 
 3   Type      804 non-null    object 
 4   Cylinder  804 non-null    int64  
 5   Liter     804 non-null    float64
 6   Doors     804 non-null    int64  
 7   Cruise    804 non-null    int64  
 8   Sound     804 non-null    int64  
 9   Leather   804 non-null    int64  
dtypes: float64(2), int64(6), object(2)
memory usage: 62.9+ KB
```

```
# 1 Produce a grid of scatterplots using only Price, Mileage, and Liter
# Note: use a semicolon after your last plot statement to supress
# the non-graphical output.

# YOUR CODE HERE
sns.pairplot(df[['Price', 'Mileage', 'Liter']], diag_kws={'bins': 10})
```

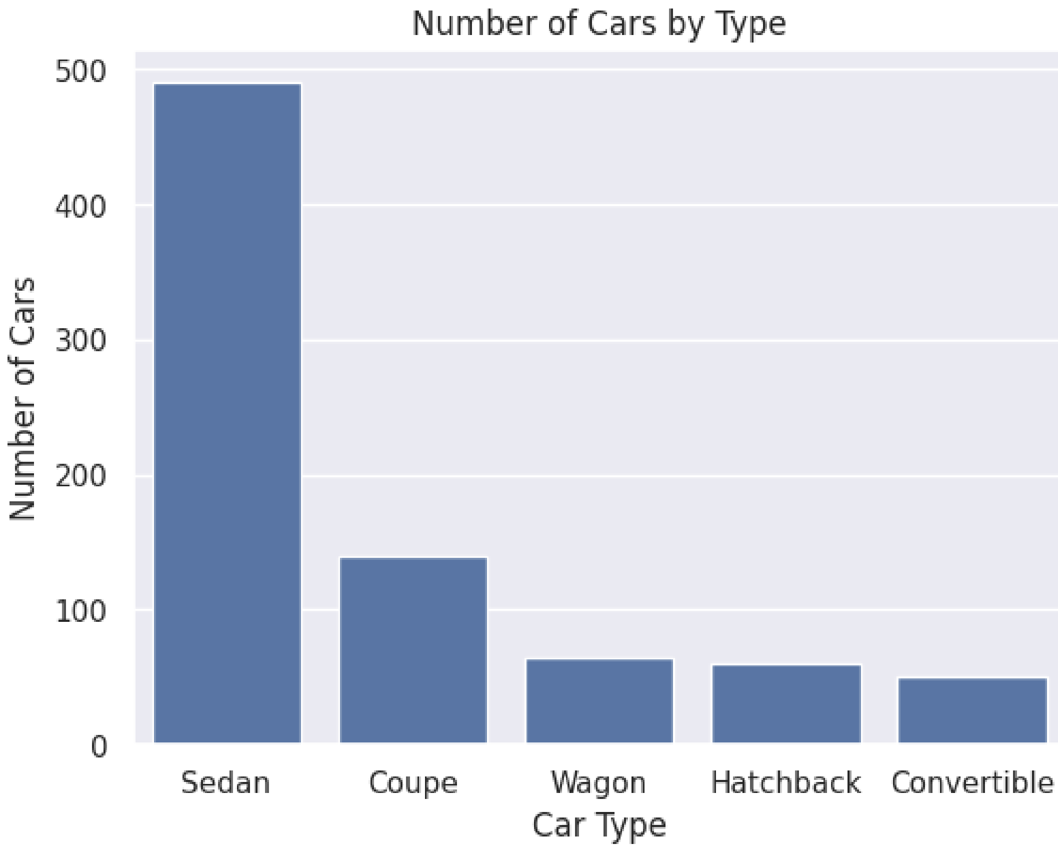
<seaborn.axisgrid.PairGrid at 0x7c79d35cc940>

<seaborn.axisgrid.PairGrid at 0x7c79d35cc940>



```
# YOUR CODE HERE
car_counts = df['Type'].value_counts()

# Create a bar plot
#plt.figure(figsize=(10, 8))
sns.barplot(x=car_counts.index, y=car_counts.values)
plt.title('Number of Cars by Type')
plt.xlabel('Car Type')
plt.ylabel('Number of Cars')
```

```
Text(0, 0.5, 'Number of Cars')
```



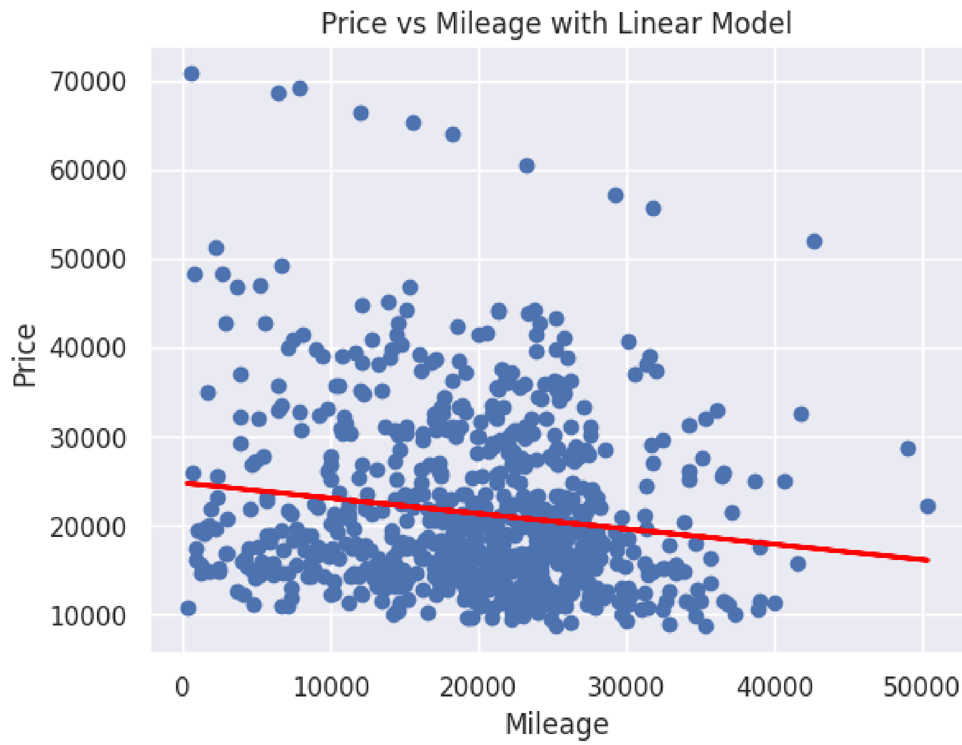
Let's build a model to predict a used car's price from its mileage.

```python
#@ 3 Build a linear model using Mileage as the single predictor variable,
# and Price as the target variable.  Fit the model using the entire data set.
# Use the LinearRegression class and use variable 'reg' for your LinearRegression object.

# YOUR CODE HERE
X = df[['Mileage']]
y = df['Price']

reg = LinearRegression()

reg.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```python
#@ 4 Create a scatterplot of Price by Mileage (Price on y axis),
# then superimpose your linear model on it (as a line).  Do not use
# Seaborn's regplot for this -- use two plotting statements.

# YOUR CODE HERE
plt.scatter(df['Mileage'], df['Price'], label='Data')

plt.plot(df['Mileage'], reg.predict(X), color='red', linewidth=2, label='Linear Model')

plt.title('Price vs Mileage with Linear Model')
plt.xlabel('Mileage')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
```



```python
#@ 5 Print the coefficients and R-squared value of the model.
# Hint: to get the R-squared value you can use the score()
# method of LinearRegression.

# YOUR CODE HERE
coefficients = reg.coef_
intercept = reg.intercept_
r_squared = reg.score(X, y)

print(f"intercept: {intercept:.2f}")
print(f"coefficients for Mileage: {coefficients[0]:.2f}")
print(f"r-squared value: {r_squared:.2f}")
```

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

```python
#@ 6 Create a new linear model for Price using predictors Mileage,
# cruise, and Leather.  Assign your model to variable reg2.

# YOUR CODE HERE
X = df[['Mileage', 'Cruise', 'Leather']]

y = df['Price']

reg2 = LinearRegression()

reg2.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```python
#@ 7 Print the coefficients (including intercept) for the new model.
# Hint: (Use print()'{:.2f}'.format(x)) to print a value
# x with 2 digits after the decimal point.)

# YOUR CODE HERE
coefficients2 = reg2.coef_
intercept2 = reg2.intercept_

print(f"Intercept: {intercept2:.2f}")
print("Coefficient:")
print(f"  Mileage: {coefficients2[0]:.2f}")
print(f"  for Cruise: {coefficients2[1]:.2f}")
print(f"  for Leather: {coefficients2[2]:.2f}")
```

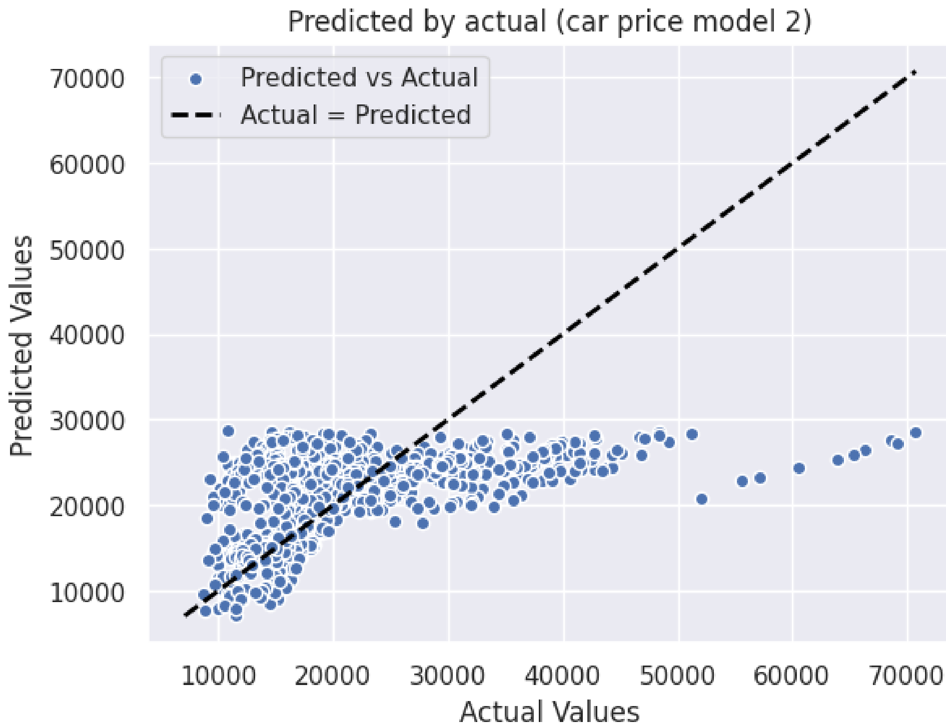
```
Intercept: 14297.18
Coefficient:
  Mileage: -0.19
  for Cruise: 10256.12
  for Leather: 4175.58
```

```python
#@ 8 Using your model, compute the predicted price for
# a car with a mileage of 20,000 and cruise but no leather.
# Hint: create a matrix to predict() that contains only one row.
# Print the predicted price.

# YOUR CODE HERE
car_mil_20k_c = [[20000, 1, 0]]

predicted_price = reg2.predict(car_mil_20k_c)

print(f"{predicted_price[0]:.2f}")
```

```
20827.73
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

```python
#@ 9 Define a function named 'plot_actual_predicted' to plot predicted vs actual values.
# It should take as input a NumPy array of actual values, a NumPy array of
# predicted values, and a plot title.  The two arrays can be assumed to be the same length.
# I used linewidth=2 and linestyle='dashed' for the actual=predicted line.
#
# Hint: when plotting, first plot the scatter plot and then plot the line that
# shows when predicted=actual.  You need only two points to plot the line where
# actual = predicted, and the points should have the form (a,a), (b,b).
# To determine what a and b should be, you may want to compute 1) the minimum
# value of actual and predicted, and 2) the maximum value of actual and predicted.

def plot_actual_predicted(actual, predicted, title):
    # YOUR CODE HERE
    plt.scatter(actual, predicted, edgecolor='white', label='Predicted vs Actual')

    min_val = min(np.min(actual), np.min(predicted))
    max_val = max(np.max(actual), np.max(predicted))

    plt.plot([min_val, max_val], [min_val, max_val], color='black', linestyle='dashed', linewidth=2, label='Actual = Predicted')

    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(title)
    plt.legend()
```

```python
# plot predicted vs. actual for your linear model

plot_actual_predicted(y, reg2.predict(X), 'Predicted by actual (car price model 2)')
```



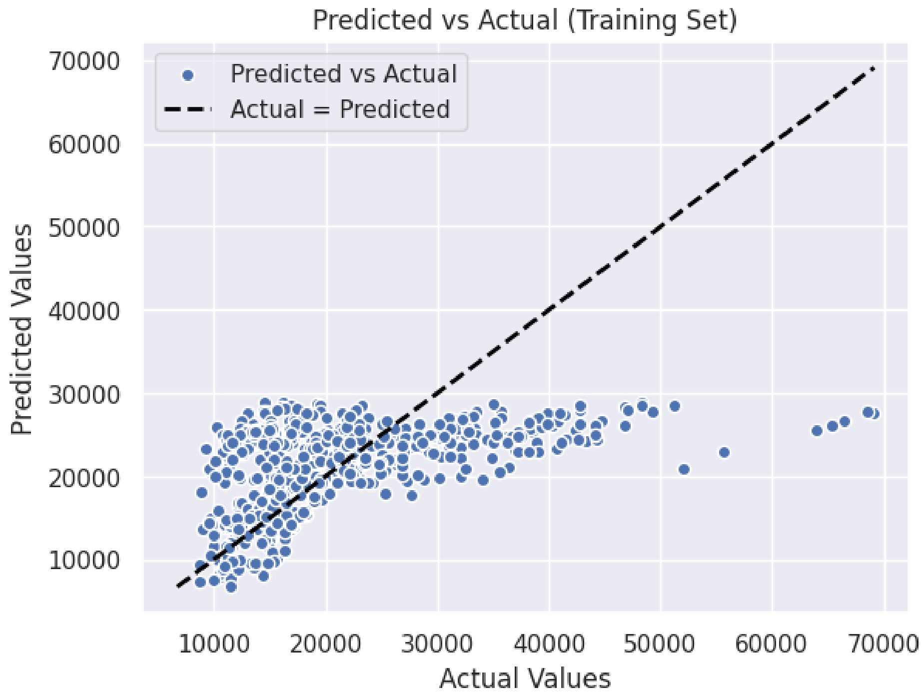

```python
[17] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

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

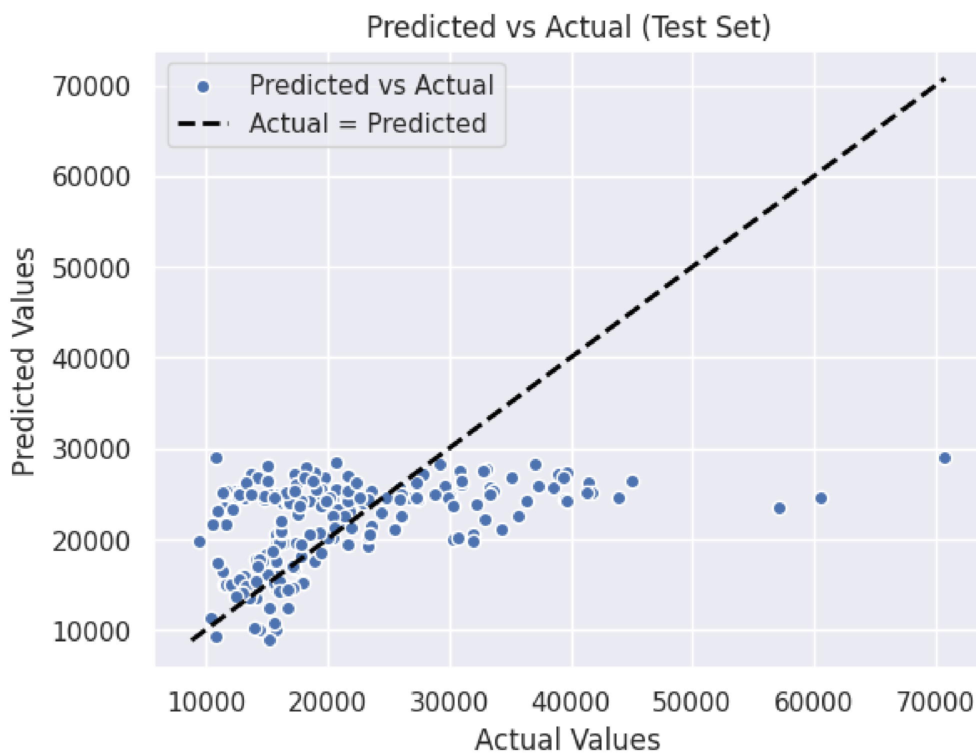
```python
#@ 10 Create another linear model (again building a model to predict Price from
# Mileage, Cruise, and Leather).  Call your new model reg3.
# However, this time fit the model using the training data.

# YOUR CODE HERE
reg3 = LinearRegression()

reg3.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
# Use the training data.

# YOUR CODE HERE
plot_actual_predicted(y_train, reg3.predict(X_train), 'Predicted vs Actual (Training Set)')
```



```python
#@ 12 Plot the actual and predicted values using plot_actual_predicted().
# This time use the test data.

# YOUR CODE HERE
plot_actual_predicted(y_test, reg3.predict(X_test), 'Predicted vs Actual (Test Set)')
```




```python
#@ 13 Print the root mean squared error on the test data.  This is the
# square root of the average squared error.  Write your own code
# to compute the RMSE; don't use a library function.

# YOUR CODE HERE
y_test_pred = reg3.predict(X_test)

squared_differences = (y_test - y_test_pred) ** 2

mean_squared_error = np.mean(squared_differences)

rmse = np.sqrt(mean_squared_error)

print(f"RMSE: {rmse:.2f}")
```

```
RMSE: 8517.23
```

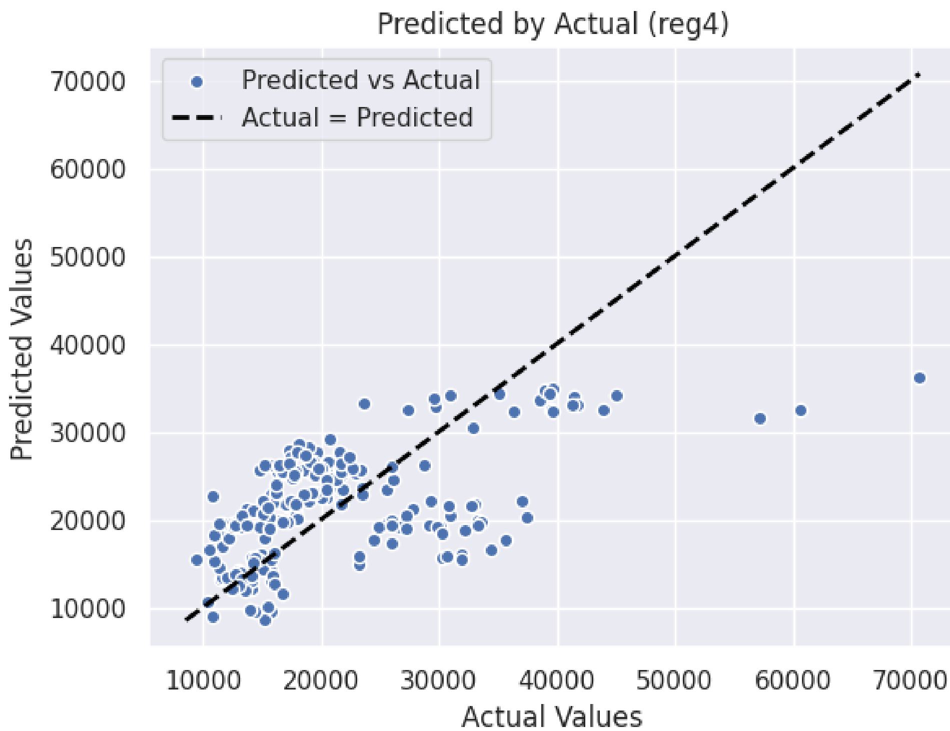
```python
#@ 14 Create a new model reg4 that is like reg3, but adds 'Cylinder'
# as a new predictor.  Do a train/test split (with random_state = 0), and fit your model to
# the training data.

# YOUR CODE HERE
X_new = df[['Mileage', 'Cruise', 'Leather', 'Cylinder']]

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.25, random_state=0)

reg4 = LinearRegression()

reg4.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
# Print the RMSE of your new model on the test data,
# and R-squared value of your new model (on the training data)

y_test_pred_reg4 = reg4.predict(X_test)
sqr_diff = (y_test - y_test_pred_reg4) ** 2
mean_squared_error = np.mean(sqr_diff)
rmse = np.sqrt(mean_squared_error)

#print('RMSE of reg4: {:.2f}'.format(np.sqrt(mean_squared_error(y_test, reg4.predict(X_test))))) #TypeError: 'numpy.float64' object is not callable
print('RMSE of reg4: {:.2f}'.format(rmse))
print('r-squared value of reg4: {:.4f}'.format(reg4.score(X_train,y_train)))
```

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

```
#@ 15 Plot the actual and predicted values using plot_actual_predicted().
# Use test data for your predictions.

# YOUR CODE HERE
plot_actual_predicted(y_test, y_test_pred_reg4, 'Predicted by Actual (reg4)')
```



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

```
#@ 16  Using predictors Mileage, Cruise, Leather, and Cylinder, make
# NumPy arrays X and y, where y contains the values in column Price.
# then scale all columns of array X using scipy.stats.zscore.
# Use X_s as the name of the scaled version of X.

# YOUR CODE HERE
X = df[['Mileage', 'Cruise', 'Leather', 'Cylinder']].values
y = df['Price'].values
X_s = zscore(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_s, y, test_size=0.25, random_state=0)

regs = LinearRegression()
regs.fit(X_train, y_train)

print('r-squared value of regs: {:.4f}'.format(regs.score(X_train,y_train)))
```

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

[27]

```
# so that, if the categorical variable has n different unique
# values, then only n-1 dummy variables will be used.

# YOUR CODE HERE
df = pd.get_dummies(df, drop_first=True)
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Price           804 non-null    float64
 1   Mileage         804 non-null    int64  
 2   Cylinder        804 non-null    int64  
 3   Liter           804 non-null    float64
 4   Doors           804 non-null    int64  
 5   Cruise          804 non-null    int64  
 6   Sound           804 non-null    int64  
 7   Leather         804 non-null    int64  
 8   Make_Cadillac   804 non-null    bool   
 9   Make_Chevrolet  804 non-null    bool   
 10  Make_Pontiac    804 non-null    bool   
 11  Make_SAAB       804 non-null    bool   
 12  Make_Saturn     804 non-null    bool   
 13  Type_Coupe      804 non-null    bool   
 14  Type_Hatchback  804 non-null    bool   
 15  Type_Sedan      804 non-null    bool   
 16  Type_Wagon      804 non-null    bool   
dtypes: bool(9), float64(2), int64(6)
memory usage: 57.4 KB
```

```python
# YOUR CODE HERE
y = df['Price'].values
X = df.drop('Price', axis=1).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

reg5 = LinearRegression()
reg5.fit(X_train, y_train)

print(f"Intercept: {reg5.intercept_:.2f}")
print("Coefficients:")
for feature, coef in zip(df.drop('Price', axis=1).columns, reg5.coef_):
    print(f"  {feature}: {coef:.2f}")
```

```
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
```

```python
#@ 19 Print the r-squared value for your model based on the training data
# and also print the RMSE based on the test data.

# YOUR CODE HERE
r_squared_train = reg5.score(X_train, y_train)
print(f"R-squared: {r_squared_train:.2f}")

y_test_pred_reg5 = reg5.predict(X_test)
sqr_diff = (y_test - y_test_pred_reg5) ** 2
mean_squared_error = np.mean(sqr_diff)
rmse = np.sqrt(mean_squared_error)

print(f"RMSE: {rmse:.2f}")
```

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

```python
#@ 20 From your NumPy array X create an extended data set X_poly using
# PolynomialFeatures with degree=2.  Assign your PolynomialFeatures
# object to variable pf.

# YOUR CODE HERE
pf = PolynomialFeatures(degree=2)

X_poly = pf.fit_transform(X)

print(X.shape)
```

```
(804, 16)
```

```python
X_poly.shape
```

```
(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?

```python
#@ 21 Create a model reg6 using all of these features.  First
# create training and test sets (using X_poly and y), then
# train a linear model on the training data, and then compute
# the RMSE on the test data.

# YOUR CODE HERE
X_train_poly, X_test_poly, y_train, y_test = train_test_split(X_poly, y, test_size=0.25, random_state=0)

reg6 = LinearRegression()
reg6.fit(X_train_poly, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
#@ 22 Using the ideas in the last cell, compute the RMSE for each individual
# feature using 5-fold cross validation.  Save the index and RMSE associated
# with the best feature as the two variables i_min and rmse_min.

# YOUR CODE HERE
#i_min = None
#rmse_min = float('inf')
#
#for i in range(X.shape[1]):
#    X_single_feature = X[:, i].reshape(-1, 1)
#    X_train, X_test, y_train, y_test = train_test_split(X_single_feature, y, te
#
#    model = LinearRegression()
#    model.fit(X_train, y_train)
#
#    y_test_pred = model.predict(X_test)
#    sqr_diff = (y_test - y_test_pred) ** 2
#    mean_squared_error_val = np.mean(sqr_diff)
#    rmse = np.sqrt(mean_squared_error_val)
#
#    print(f"Feature index {i} RMSE: {rmse:.2f}")
#
#    if rmse < rmse_min:
#        rmse_min = rmse
#        i_min = i

#print(f"num features: {i_min}")
#print(f"Minimum RMSE: {rmse_min:.2f}")

def get_rmse(model, X, y, n_splits=5):
    n_samples = len(y)
    indices = np.arange(n_samples)
    np.random.shuffle(indices)
    fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=int)
    fold_sizes[:n_samples % n_splits] += 1
    current = 0
```

```python
    rmse_values = []

    for fold_size in fold_sizes:
        start, stop = current, current + fold_size
        test_indices = indices[start:stop]
        train_indices = np.concatenate((indices[:start], indices[stop:]))
        current = stop

        X_train, X_test = X[train_indices], X[test_indices]
        y_train, y_test = y[train_indices], y[test_indices]

        model.fit(X_train, y_train)
        y_test_pred = model.predict(X_test)
        sqr_diff = (y_test - y_test_pred) ** 2
        mean_squared_error_val = np.mean(sqr_diff)
        rmse = np.sqrt(mean_squared_error_val)
        rmse_values.append(rmse)

    return np.mean(rmse_values)

i_min = None
rmse_min = float('inf')

for i in range(X.shape[1]):
    X_single_feature = X[:, i].reshape(-1, 1)
    model = LinearRegression()
    rmse = get_rmse(model, X_single_feature, y)
    print(f"Feature index {i} RMSE: {rmse:.2f}")

    if rmse < rmse_min:
        rmse_min = rmse
        i_min = i
```

```
Feature index 0 RMSE: 9786.02
Feature index 1 RMSE: 8083.76
Feature index 2 RMSE: 8187.80
Feature index 3 RMSE: 9771.89
Feature index 4 RMSE: 8902.98
Feature index 5 RMSE: 9787.09
Feature index 6 RMSE: 9751.17
Feature index 7 RMSE: 7436.61
Feature index 8 RMSE: 9013.57
Feature index 9 RMSE: 9779.46
Feature index 10 RMSE: 9280.63
Feature index 11 RMSE: 9644.46
Feature index 12 RMSE: 9740.64
Feature index 13 RMSE: 9655.63
Feature index 14 RMSE: 9886.52
Feature index 15 RMSE: 9865.82
```

```python
# make a version of the training data with just feature 0
X_0 = X_train[:,[0]]

# compute negated mean square error scores using 5-fold cross validation
scores = cross_val_score(LinearRegression(), X_0, y_train, scoring='neg_mean_squared_error', cv=5)

# work out the average root mean squared error.  We need to
# first negate the scores, because they are negative MSE, not MSE.
rmse = np.sqrt(-scores.mean())

print('RMSE for feature 0 only: {:.2f}'. format(rmse))
```

```
RMSE for feature 0 only: 9882.58
```

```python
#print('best feature: {}, best RMSE: {:.2f}'.format(pf.get_feature_names()[i_min], rmse_min))
print('best feature: {}, best RMSE: {:.2f}'.format(pf.get_feature_names_out()[i_min], rmse_min))
```

```
best feature: x6, best RMSE: 7436.61
```

```python
#@ 23 Find the 10 features that give the lowest RMSE by using
# forward search.  Important: find the single best feature, then
# find one more feature that is the best *when combined with
# the single best feature*, and continue, always looking for
# the single best feature combined with all previously-selected
# features.
#
# I've filled in some of the code for you.  'remaining' is a list
# of the features to be considered when finding the next best
# feature.  It is initialized to all the features.  'selected' is
# a list of the features that have been chosen in a round of finding the
# next best feature.  It is initialized to the empty list.
#
# Don't forget to have include all selected features when looking
# for the next best feature.

remaining = list(range(X_train.shape[1]))
selected = []
n = 10
while len(selected) < n:
    # find the single features that works best in conjunction
    # with the already selected features
    rmse_min = 1e7
    for i in remaining:
        # YOUR CODE HERE
        X_selected = np.concatenate([X_train[:, selected], X_train[:, [i]]], axis=1)
        model = LinearRegression()
        model.fit(X_selected, y_train)
        rmse = get_rmse(model, X_selected, y_train)
        if rmse < rmse_min:
            rmse_min = rmse
            i_min2 = i


    remaining.remove(i_min2)
    selected.append(i_min2)
    print('num features: {}; rmse: {:.2f}'.format(len(selected), rmse_min))
```

```
num features: 1; rmse: 7359.10
num features: 2; rmse: 5995.09
num features: 3; rmse: 3990.66
num features: 4; rmse: 3636.67
num features: 5; rmse: 3456.31
num features: 6; rmse: 3357.37
num features: 7; rmse: 2692.68
num features: 8; rmse: 2544.02
num features: 9; rmse: 2520.73
num features: 10; rmse: 2533.02
```

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

```
#@ 24 Print the RMSE of the best 10 features on the test set

# YOUR CODE HERE
X_train_selected = X_train[:, selected]
X_test_selected = X_test[:, selected]

reg_selected = LinearRegression()
reg_selected.fit(X_train_selected, y_train)

y_test_pred_selected = reg_selected.predict(X_test_selected)

rmse_selected = get_rmse(reg_selected, X_test_selected, y_test)

print("RMSE of the model with the best 10 features:", rmse_selected)
```

```
RMSE of the model with the best 10 features: 2848.202821302465
```