Boston Housing Data

In order to gain a better understanding of the metrics used in regression settings, we will be looking at the Boston Housing dataset.

First use the cell below to read in the dataset and set up the training and testing data that will be used for the rest of this problem.

In [1]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import numpy as np
import tests2 as t

boston = load_boston()
y = boston.target
X = boston.data

X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.33, random_state=42)
```

Step 1: Before we get too far, let's do a quick check of the models that you can use in this situation given that you are working on a regression problem. Use the dictionary and corresponding letters below to provide all the possible models you might choose to use.

In [2]:
```
# When can you use the model - use each option as many times as necessary
a = 'regression'
b = 'classification'
c = 'both regression and classification'

models = {
    'decision trees': c,
    'random forest': c,
    'adaptive boosting': c,
    'logistic regression': b,
    'linear regression': a,
}

#checks your answer, no need to change this code
t.q1_check(models)
```

```
That's right!  All but logistic regression can be used for predicting numeric values.  And linear regr
ession is the only one of these that you should not use for predicting categories.  Technically sklear
n won't stop you from doing most of anything you want, but you probably want to treat cases in the way
you found by answering this question!
```

> **Step 2:** Now for each of the models you found in the previous question that can be used for regression problems, import them using sklearn.

In [3]:
```
# Import models from sklearn - notice you will want to use
# the regressor version (not classifier) - googling to find
# each of these is what we all do!

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

> **Step 3:** Now that you have imported the 4 models that can be used for regression problems, instantate each below.

In [4]:
```
# Instantiate each of the models you imported
# For now use the defaults for all the hyperparameters

tree_mod = DecisionTreeRegressor()
rf_mod = RandomForestRegressor()
ada_mod = AdaBoostRegressor()
reg_mod = LinearRegression()
```

Step 4: Fit each of your instantiated models on the training data.

In [5]:
```
# Fit each of your models using the training data
# Fit each of your models using the training data
tree_mod.fit(X_train, y_train)
rf_mod.fit(X_train, y_train)
ada_mod.fit(X_train, y_train)
reg_mod.fit(X_train, y_train)
```

Out[5]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Step 5: Use each of your models to predict on the test data.

In [6]:
```
# Predict on the test values for each model

preds_tree = tree_mod.predict(X_test)
preds_rf = rf_mod.predict(X_test)
preds_ada = ada_mod.predict(X_test)
preds_reg = reg_mod.predict(X_test)
```

Step 6: Now for the information related to this lesson. Use the dictionary to match the metrics that are used for regression and those that are for classification.

In [7]:
```
# potential model options
a = 'regression'
b = 'classification'
c = 'both regression and classification'

#
metrics = {
    'precision': b,
    'recall': b,
    'accuracy': b,
    'r2_score': a,
    'mean_squared_error': a,
    'area_under_curve': b,
    'mean_absolute_area': a
}

#checks your answer, no need to change this code
t.q6_check(metrics)
```

```
That's right! Looks like you know your metrics!
```

Step 6: Now that you have identified the metrics that can be used in for regression problems, use sklearn to import them.

In [8]:
```
# Import the metrics from sklearn
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

Step 7: Similar to what you did with classification models, let's make sure you are comfortable with how exactly each of these metrics is being calculated. We can then match the value to what sklearn provides.

In [9]:
```
def r2(actual, preds):
    '''
    INPUT:
    actual - numpy array or pd series of actual y values
    preds - numpy array or pd series of predicted y values
    OUTPUT:
    returns the r-squared score as a float
    '''
    sse = np.sum((actual-preds)**2)
    sst = np.sum((actual-np.mean(actual))**2)
    return 1 - sse/sst

# Check solution matches sklearn
print(r2(y_test, preds_tree))
print(r2_score(y_test, preds_tree))
print("Since the above match, we can see that we have correctly calculated the r2 value.")
```

```
0.721991330305
0.721991330305
Since the above match, we can see that we have correctly calculated the r2 value.
```

Step 8: Your turn fill in the function below and see if your result matches the built in for mean_squared_error.

In [10]:
```
def mse(actual, preds):
    '''
    INPUT:
    actual - numpy array or pd series of actual y values
    preds - numpy array or pd series of predicted y values
    OUTPUT:
    returns the mean squared error as a float
    '''

    return np.sum((actual-preds)**2)/len(actual)


# Check your solution matches sklearn
print(mse(y_test, preds_tree))
print(mean_squared_error(y_test, preds_tree))
print("If the above match, you are all set!")
```

```
21.0392814371
21.0392814371
If the above match, you are all set!
```

Step 9: Now one last time - complete the function related to mean absolute error. Then check your function against the sklearn metric to assure they match.

In [11]:
```
def mae(actual, preds):
    '''
    INPUT:
    actual - numpy array or pd series of actual y values
    preds - numpy array or pd series of predicted y values
    OUTPUT:
    returns the mean absolute error as a float
    '''

    return np.sum(np.abs(actual-preds))/len(actual)

# Check your solution matches sklearn
print(mae(y_test, preds_tree))
print(mean_absolute_error(y_test, preds_tree))
print("If the above match, you are all set!")
```

```
3.2874251497
3.2874251497
If the above match, you are all set!
```

Step 10: Which model performed the best in terms of each of the metrics? Note that r2 and mse will always match, but the mae may give a different best model. Use the dictionary and space below to match the best model via each metric.

In [12]:
```
#match each metric to the model that performed best on it
a = 'decision tree'
b = 'random forest'
c = 'adaptive boosting'
d = 'linear regression'


best_fit = {
    'mse': b,
    'r2': b,
    'mae': b
}

#Tests your answer - don't change this code
t.check_ten(best_fit)
```

```
That's right!  The random forest was best in terms of all the metrics this time!
```

In [ ]:
```
# cells for work
```

In [13]:
```
def print_metrics(y_true, preds, model_name=None):
    '''
    INPUT:
    y_true - the y values that are actually true in the dataset (numpy array or pandas series)
    preds - the predictions for those values from some model (numpy array or pandas series)
    model_name - (str - optional) a name associated with the model if you would like to add it to the p

    OUTPUT:
    None - prints the mse, mae, r2
    '''
    if model_name == None:
        print('Mean Squared Error: ', format(mean_squared_error(y_true, preds)))
        print('Mean Absolute Error: ', format(mean_absolute_error(y_true, preds)))
        print('R2 Score: ', format(r2_score(y_true, preds)))
        print('\n\n')

    else:
        print('Mean Squared Error ' + model_name + ' :' , format(mean_squared_error(y_true, preds)))
        print('Mean Absolute Error ' + model_name + ' :', format(mean_absolute_error(y_true, preds)))
        print('R2 Score ' + model_name + ' :', format(r2_score(y_true, preds)))
        print('\n\n')
```

In [14]:
```
# Print Decision Tree scores
print_metrics(y_test, preds_tree, 'tree')

# Print Random Forest scores
print_metrics(y_test, preds_rf, 'random forest')

# Print AdaBoost scores
print_metrics(y_test, preds_ada, 'adaboost')

# Linear Regression scores
print_metrics(y_test, preds_reg, 'linear reg')
```

```
Mean Squared Error tree : 21.039281437125748
Mean Absolute Error tree : 3.287425149700599
R2 Score tree : 0.7219913303051109



Mean Squared Error random forest : 15.199392814371258
Mean Absolute Error random forest : 2.5171257485029943
R2 Score random forest : 0.7991583985830897



Mean Squared Error adaboost : 16.31417907934795
Mean Absolute Error adaboost : 2.838626338203197
R2 Score adaboost : 0.7844278457623349



Mean Squared Error linear reg : 20.747143360309067
Mean Absolute Error linear reg : 3.1512878365884154
R2 Score linear reg : 0.7258515818230032
```

In [ ]:
```

```