Recap

You've built a model. In this exercise you will test how good your model is.

Run the cell below to set up your coding environment where the previous exercise left off.

In [1]:
```
# Code you have previously used to load data
import pandas as pd
from sklearn.tree import DecisionTreeRegressor

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
y = home_data.SalePrice
feature_columns = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[feature_columns]

# Specify Model
iowa_model = DecisionTreeRegressor()
# Fit Model
iowa_model.fit(X, y)

print("First in-sample predictions:", iowa_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex4 import *
print("Setup Complete")
```

```
First in-sample predictions: [208500. 181500. 223500. 140000. 250000.]
Actual target values for those homes: [208500, 181500, 223500, 140000, 250000]
Setup Complete
```

Exercises

Step 1: Split Your Data

Use the `train_test_split` function to split up your data.

Give it the argument `random_state=1` so the `check` functions know what to expect when verifying your code.

Recall, your features are loaded in the DataFrame **X** and your target is loaded in **y**.

In [2]:
```
# Import the train_test_split function
from sklearn.model_selection import train_test_split

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 1)

step_1.check()
```

Correct

In [3]:
```
# The lines below will show you a hint or the solution.
step_1.hint()
step_1.solution()
```

Hint: The function you need to import is part of sklearn. When calling the function, the arguments are X and y. Ensure you set the random_state to 1.

Solution:
```
from sklearn.model_selection import train_test_split
train_x, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
```

Step 2: Specify and Fit the Model

Create a `DecisionTreeRegressor` model and fit it to the relevant data. Set `random_state` to 1 again when creating the model.

In [4]:
```
# You imported DecisionTreeRegressor in your last exercise
# and that code has been copied to the setup code above. So, no need to
# import it again

# Specify the model
iowa_model = DecisionTreeRegressor(random_state=1)

# Fit iowa_model with the training data.
iowa_model.fit(train_X, train_y)
step_2.check()
```

```
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
```

Correct

In [5]:
```
step_2.hint()
step_2.solution()
```

Hint: Remember, you fit with training data. You will test with validation data soon

Solution:
```
iowa_model = DecisionTreeRegressor(random_state=1)
iowa_model.fit(train_X, train_y)
```

Step 3: Make Predictions with Validation data

In [6]:
```
# Predict with all validation observations
val_predictions = iowa_model.predict(val_X)
step_3.check()
```

Correct

In [7]:
```
step_3.hint()
step_3.solution()
```

Hint: Run predict on the right validation data object.

Solution:
```
val_predictions = iowa_model.predict(val_X)
```

Inspect your predictions and actual values from validation data.

In [8]:
```
# print the top few validation predictions
print(val_predictions)
# print the top few actual prices from validation data
print(val_y)
```

```
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000. 180000. 121000. 175900. 210000. 248900. 131000. 100000. 149350.
 235000. 156000. 149900. 265979. 193500. 377500. 100000. 162900. 145000.
 180000. 582933. 146000. 140000.  91500. 112500. 113000. 145000. 312500.
 110000. 132000. 305000. 128000. 162900. 115000. 110000. 124000. 215200.
 180000.  79000. 192000. 282922. 235000. 132000. 325000.  80000. 237000.
 208300. 100000. 120500. 162000. 153000. 187000. 185750. 335000. 129000.
 124900. 185750. 133700. 127000. 230000. 146800. 157900. 136000. 153575.
 335000. 177500. 143000. 202500. 168500. 105000. 305900. 192000. 190000.
 140200. 134900. 128950. 213000. 108959. 149500. 190000. 175900. 160000.
 250580. 157000. 120500. 147500. 118000. 117000. 110000. 130000. 148500.
 148000. 190000. 130500. 127000. 120500. 135000. 168000. 176432. 128000.
 147000. 260000. 132000. 129500. 171000. 181134. 227875. 189000. 282922.
  94750. 185000. 194000. 159000. 279500. 290000. 135000. 299800. 165000.
 394432. 135750. 155000. 212000. 310000. 134800.  84000. 122900.  80000.
 191000. 755000. 147000. 248000. 106500. 145000. 359100. 145000. 192500.
 149000. 252000. 109000. 215000. 220000. 138500. 185000. 185000. 120500.
 181000. 173000. 335000.  67000. 149350.  67000. 156000. 119000. 110500.
 184000. 147000. 156000. 171000. 177000. 159000. 125000. 105000. 284000.
 167500. 200000. 312500. 213000. 135960. 205000. 237000. 107000. 163000.
 132500. 155835. 165500. 138500. 257000. 160000. 394617. 281213. 161000.
 127500.  88000. 139000.  89500. 132500. 134800. 335000. 248900. 155000.
 147000.  86000. 185000. 200000. 180500. 215200. 319900. 105000. 194000.
 340000. 256000. 280000. 186500. 105500. 155000. 133500. 255500. 253000.
 130000.  92900. 256000. 100000. 755000. 138500. 168500. 112000. 127000.
 109008. 197000. 245500. 171900. 162000. 128000. 173000. 132000. 118000.
 235128. 118964. 260000. 116000. 185000. 315750. 236500. 140000. 151500.
 184000.  84000. 130000. 154000. 205000. 110000. 151500. 123000. 129500.
 173900. 181500. 165500. 106500. 184900.  84500. 377500. 118500. 180000.
 190000. 208500. 181000.  98000. 157000. 151500.  84000. 139000. 100000.
 161750. 165600. 116000. 118500. 187000. 147000. 112000. 132000. 230000.
 128000. 147000. 125000. 145000. 151000. 284000. 221000. 140200. 129000.
 290000. 105000.  96500. 310000. 140000. 132000. 203000. 221000. 215200.
 214000. 139000.  91500. 148000. 155000. 115000. 180000. 165500. 223000.
 139000. 179900. 150000. 185000. 163000. 176000. 127000. 227000. 146000.
  99900. 275000. 180500. 180000. 157000. 186500. 179900. 137500. 219500.
 155000. 345000. 197000. 205000. 159000. 159434. 156000. 196000. 252678.
 255500. 213000. 150900. 143750. 139000. 260000. 189000. 213250. 207500.
  80000. 221000. 109500. 155000. 165000. 149350. 204900. 105900. 155000.
 176000. 395000. 149700. 147000. 143900. 226700. 176000. 116000. 325300.
 133750. 188500. 148500. 284000. 201800.]
258     231500
267     179500
288     122000
649      84500
1233    142000
167     325624
926     285000
831     151000
1237    195000
426     275000
487     175000
375      61000
1126    174000
53      385000
1033    230000
1022     87000
1215    125000
91       98600
1270    260000
680     143000
464     124000
1416    122500
730     236500
994     337500
383      76000
992     187000
531     128000
742     179000
798     485000
432     122500
         ...
1003    136905
126     128000
1206    107000
718     341000
1195    176000
815     224900
503     289000
935      79900
640     274000
90      109900
925     175000
830     166000
262     151000
765     264132
```

```
589      79500
399     241000
148     141000
336     377426
368     132000
556     141000
78      136500
650     205950
880     157000
821      93000
35      309000
1017    187500
534     178000
1334    125000
1369    232000
628     135000
Name: SalePrice, Length: 365, dtype: int64
```

What do you notice that is different from what you saw with in-sample predictions (which are printed after the top code cell in this page).

Do you remember why validation predictions differ from in-sample (or training) predictions? This is an important idea from the last lesson.

Step 4: Calculate the Mean Absolute Error in Validation Data

In [9]:
```python
from sklearn.metrics import mean_absolute_error
val_mae = mean_absolute_error(val_y, val_predictions)

# uncomment following line to see the validation_mae
print(val_mae)
step_4.check()
```

```
29652.931506849316
```

Correct

In [10]:
```python
step_4.hint()
step_4.solution()
```

Hint: The order of arguments to mean_absolute_error doesn't matter. Make sure you fit to only the training data in step 2.

Solution:
```python
val_mae = mean_absolute_error(val_predictions, val_y)
```

Is that MAE good? There isn't a general rule for what values are good that applies across applications. But you'll see how to use this number in the next step.

Keep Going

Now that you can measure model performance, you are ready to run some experiments comparing different models. The key is to understand **Underfitting and Overfitting**. It's an especially fun part of machine learning.

Course Home Page