This notebook is an exercise in the [Introduction to Machine Learning](#) course. You can reference the tutorial at [this link](#).

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 [ ]:

```
# 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', 'Ful
lBath', '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")
```

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 [ ]:
```
# Import the train_test_split function and uncomment
from sklearn.model_selection import train_test_split

# fill in and uncomment
train_X, val_X, train_y, val_y = train_test_split(X,y,random_state = 1)

step_1.check()
```

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

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 [ ]:
```
# 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()
```

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

Step 3: Make Predictions with Validation data

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

step_3.check()
```

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

Inspect your predictions and actual values from validation data.

In [ ]:
```
# print the top few validation predictions
print(val_y.head())
# print the top few actual prices from validation data
print(val_X.head())
```

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 [ ]:
```
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()
```

In [ ]:
```
# step_4.hint()
# step_4.solution()
```

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 (and improve) this number in the next step.

Keep Going

You are ready for **[Underfitting and Overfitting](#)**.

Have questions or comments? Visit the [Learn Discussion forum](#) to chat with other Learners.