Introduction to Machine Learning: 3rd lesson – Your First Machine Learning Model

Selecting data for modeling:

* Your dataset had too many variables to wrap your head around, or even to print out nicely. How can you pare down this overwhelming amount of data to something you can understand?
* We'll start by picking a few variables using our intuition. Later courses will show you statistical techniques to automatically prioritize variables.
* To choose variables or columns, we'll need to see a list of all columns in the dataset. That is done with the columns property of the DataFrame (the bottom line of code below):

import pandas as pd

melbourne\_file\_path = '../input/melbourne-housing-snapshot/melb\_data.csv'

melbourne\_data = pd.read\_csv(melbourne\_file\_path)

melbourne\_data.columns

Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',

'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',

'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',

'Longtitude', 'Regionname', 'Propertycount'],

dtype='object')

# The Melbourne data has some missing values (some houses for which some variables weren't recorded.)

# We'll learn to handle missing values in a later tutorial.

# Your Iowa data doesn't have missing values in the columns you use.

# So we will take the simplest option for now, and drop houses from our data.

# Don't worry about this much for now, though the code is:

# dropna drops missing values (think of na as "not available")

melbourne\_data = melbourne\_data.dropna(axis=0)

* There are many ways to select a subset of your data. The pandas course covers these in more depth, but we will focus on two approaches for now.

1} Dot notation, which we use to select the "prediction target"

2} Selecting with a column list, which we use to select the "features"

* Selecting the prediction target:
* You can pull out a variable with dot-notation. This single column is stored in a Series, which is broadly like a DataFrame with only a single column of data.
* We'll use the dot notation to select the column we want to predict, which is called the prediction target. By convention, the prediction target is called y. So the code we need to save the house prices in the Melbourne data is:

y = melbourne\_data.Price

Choosing “features”:

* The columns that are inputted into our model (and later used to make predictions) are called "features." In our case, those would be the columns used to determine the home price. Sometimes, you will use all columns except the target as features. Other times you'll be better off with fewer features.
* For now, we'll build a model with only a few features. Later on you'll see how to iterate and compare models built with different features. We select multiple features by providing a list of column names inside brackets. Each item in that list should be a string (with quotes). Here is an example:

melbourne\_features = ['Rooms', 'Bathroom', 'Landsize', 'Latitude', 'Longitude']

* By convention, this data is called X:

X = melbourne\_data[melbourne\_features]

Building your model:

* You will use the scikit-learn library to create your models. When coding, this library is written as sklearn, as you will see in the sample code. Scikit-learn is easily the most popular library for modeling the types of data typically stored in DataFrames. The steps to building and using a model are:
* Define: What type of model will it be? A decision tree? Some other type of model? Some other parameters of the model type are specified too.
* Fit: Capture patterns from provided data. This is the heart of modeling.
* Predict: Just what it sounds like
* Evaluate: Determine how accurate the model's predictions are.
* Here is an example of defining a decision tree model with scikit-learn and fitting it with the features and target variable:

from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random\_state to ensure same results each run

melbourne\_model = DecisionTreeRegressor(random\_state=1)

# Fit model

melbourne\_model.fit(X, y)

DecisionTreeRegressor(random\_state=1)