1. How Models Work

We use data to decide how to break the houses into two groups, and then again to determine the predicted price in each group. This step of capturing patterns from data is called fitting or training the model. The data used to fit the model is called the training data.

2. Basic Data Exploration

1. Using Pandas to Get Familiar With Your Data :

The first step in any machine learning projects is familiarize yourself with the data. You'll use the Pandas library for this. Pandas is the primary tools data scientists use for exploring and manipulating data. Most people abbreviate pandas in their code as pd. The most important part of the Pandas library is the DataFrame. A DataFrame holds the type of data you might think of as a table. This is similar to a sheet in Excel, or a table in a SQL database.
Pandas has powerful methods for most things you'll want to do with this type of data.

1. Interpreting Data Description :

The results show 8 numbers for each colum┬ in your original dataset. The first number, the count, shows how many rows have non-missing values.
Missing values arise for many reasons. For example, the size of the 2nd bedroom wouldn't be collected when surveying a 1 bedroom house. We'll come back to the topic of missing data.

The second value is the mean, which is the average. Under that, std is the standard deviation, which measures how numerically spread out the values are. To interpret the min, 25%, 50%, 75% and max values, imagine sorting each column from lowest to highest value. The first value is the min. If you go to quarter way through the list, you'll find a number that is bigger than 25% of the values and smaller than 75% of the values. Thsat is the 25% value. The 50th and 75th percentiles are defined analogously, and the max is the largest number.

3. Your First Machine Learning Model

1. Selecting Data for Modeling :

Your dataset had too many variables to wrap your head around, or even to pring out nicely. How can we pare down this overwhelming amount of data to something we can understand?

We'll start by picking a few variables using our intuition. Later course will show us statistical techniques to automatically prioritize variables. To choose variables/columns, we'll need to see a list of all columns in the dataset. That is done with the column property of the DataFrame.

There are many ways to select a subset of our data. The pandas course covers these in more depth.

1. Selecting The Predicting Target :

We can pull out a vraible with dot-notation. This single column is stored in a Series, which is broadly like a DataFrame with only a single column of data.

1. Choosing 'Features' :

The columns that are inputted into our model are called 'features'. Sometimes, we will use all columns except the target as features. Other times we'll be better off with fewer features. We select multiple features by providing a list of column names inside brackets. Each item in that list should be a string. By convention, this data is called X.

1. Building Your Model :

We will use the scikit-learn library to create our modles. When coding, this library is written as sklearn, as we 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.

Many machine learning models allow some randomness in model training. Specifying a number for random_state ensures you get the same results in each run. This is considered a good practice. You use any number, and model quality won't depend meaningfully on exactly what value you choose.

```
# Loading dataset

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

# Data preprocessing

melbourne_data = melbourne_data.dropna(axis=0)

# Feature Engineering

y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
X = melbourne_data[melbourne_features]

# Modeling

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)
```

4. Model validation

1. What is Model Validation :

you'll want to evaluate almost every model you ever build. In most applications, the relevant measure of model quality is predictive accuracy. In other words, will the model's predictions be close to what actually happens.

Many people make a huge mistake when meauring predictive accuracy. They make predictions with their training data and compare those predictions to the traget values in the training data. you'll see the problem with this approach and how to solve it in a moment, but let's think about how we'd to this first.
There are many metrics for summarizing model quality, but we'll start with one called Mean Absolute Error(MAE). With the MAE metric, we take the asolute value of each error. THis converts each error to a positive number. We then take the average of those absolute errors. This is our measure of model quality.

```
# apply MAE
from sklearn.metric import mean_absolute error

predicted_home_price = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_price)
```

1. The problem with "in-sample" Score :

The measure we just computed can be called an "in-sample" score. We used a single "sample" of houses for both building the model and evaluating it. Here's why this is bad. Imagine that, in the large real estate market, door color is unrelated to home price.

However, in the sample of data you used to build the model, all homes with green doors were very expensive. The model's hob is to find patterns that predict home prices, so it will see this pattern, and it will always predict highprices for homes with green doors. Since this pattern was derived from the training dataa, the model will appear accurate in the training data. But if this pattern doesn't hold when the model sees new data, the model would be very inaccurate when used in practice.

Since models' practical value come from making predictions on new data, we measure performance on data that wasn't used to build the model. The most straightforward way to do this is to exclude some data from the model-building process, and then use those to test the model's accuracy on data it hasn't seen before. This data is called Validation data.

1. Coding it :

The scikit-learn library has a function train_test_split to break up the data into two pieces. We'll use some of that data as training data to fit the model, and we'll use the other data as validation data to calculate.

```
from sklearn.model_selection import train_test_split

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)

melbourne_modle = DecisionTreeRegressor()
melbourne_model.fit(train_X, train_y)

val_predicitons = melbourne_model.predict(val_X)
mean_absolute_error(val_y, val_predictions)
```

5. Underfitting and Overfitting

1. Experimenting With Different Models :

Now that you have a reliable way to measure model accuracy, you can experiment with alternative model and see which gives teh best predictions. But what alternative do you have for models?

You can see in scikit-learn's documentation that the decision tree model has many options. The most important options deterimine the tree's depth. In practice, it's not uncommon for a tree to have 10 splits between the top level and a leaf. As the tree gets deeper, the dataset gets sliced up into leaves with feter houses. If a tree only had 1 split. It divides the data into 2 groups. If each group is split again, we would get 4 groups of houses. Splitting each of those again would create 8 groups.

When we divide the houses amongst many levaes, we also have fewer houses in each leaf. Leaves with very few houses will make predictions that are quite close tho those homes' actual value, but they make very unreliable predictions for new data.

This is a phonomenon called overfitting, where a model matches the training data almost perfectly, but does poorly in validation and other new data. On the flip side, if we make our tree very shallow, it doen't divide up the houses into very distinct groups.

At an extreme, if a tree divdes houses into only 2 or 4, each group still has a wide variety of hoses. Resulting predictions may be far off most houses, even inthe training data. When a model fails to capture important disticntions and patterns in the data, so it performs pooly even in training data, that is called underfitting.

6. Random Forests

Decision trees leave ou with a difficult decision. A deep tree with lots of leaves will overfit because each prediction is coming from historical data from only the few houses at ist leaf. But a shallow tree with fe leaves will perform pooly because it fails to capture as many distinction in te raw data.

Even today's most sophisticated modeling techniques face this tension between underfitting and overfitting. But, many models have clever ideas that can lead to better performance.

The randomforest uses many trees, and it makes a prediction by averaging the predictions of each component tree. It generally has much better predictive accuracy than a single decision tree and it works well with default parameters. If you keep modeling, you can learn more models with even better performance, but many of those are sensitive to getting the right parameters.