**An introduction to Supervised Machine Learning in Python**

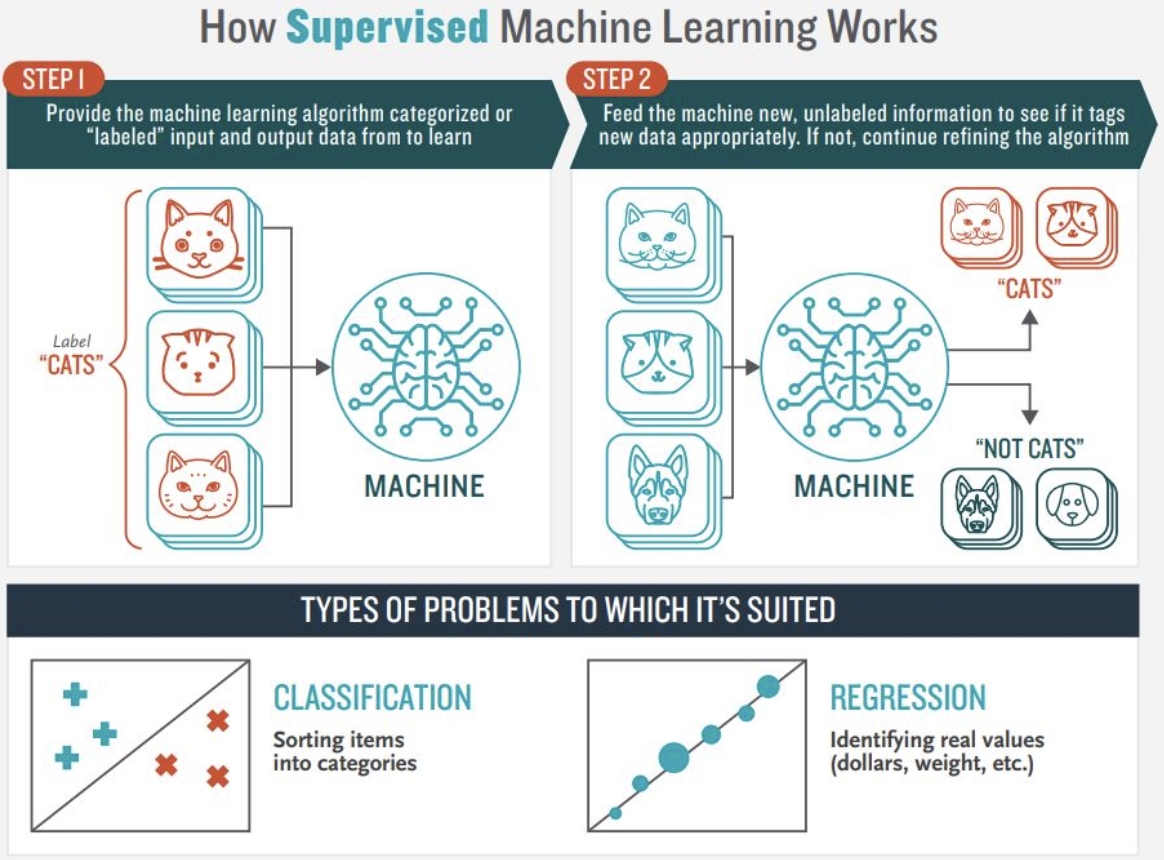
This document should serve as a brief introduction to running supervised machine learning algorithms in python. I assume that you’ve had a look at the previous documents on getting started with python and jupyter, and have looked at the pandas/data visualisation documents. Here is an academic paper with an introduction to machine learning in general, that goes through the most common types: <https://openaccess.iyte.edu.tr/bitstream/11147/5770/1/5770.pdf>

**Introduction to Supervised Machine Learning:**

Supervised machine learning involves attempting to train an algorithm to predict a value or class of new data based on “labelled data” that you train it on.

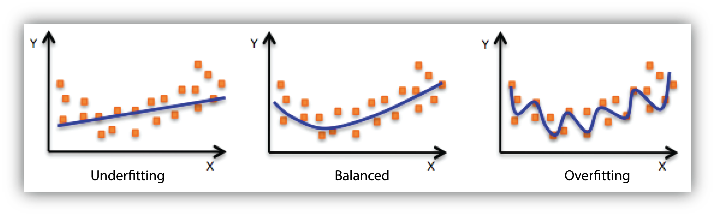
For example, if we give an algorithm a set of images that we have labelled as pictures of cats of dogs, can the algorithm learn from the images what constitutes a cat or dog picture, and therefore predict the label for any new image?

Similarly – if we give an algorithm a dataset of polymer properties, including a metric for their “stickiness”, can we predict the stickiness of a new polymer if the algorithm is given its properties?



The way these algorithms generally work is that you train the model to learn the relationships in the data, then you test it on a set of unlabelled data that it hasn’t seen before.

By scoring the algorithm and giving it a chance to change its understanding of the relationships, you allow it to become better at predictions over a series of cycles. This process is generally called “fitting”. The more we fit the model, the better it can become at predicting the data you have, but there is a balance between not enough fitting (and your model isn’t predictive) and too much fitting (your model has learnt the noise in the data rather than the “true” relationship:



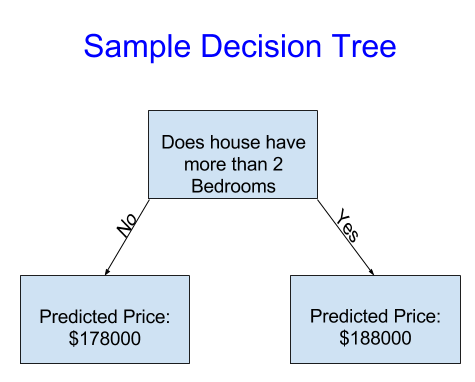
This is called underfitting and overfitting.

The process of performing supervised machine learning is generally separated into the following stages:

* Curation and cleaning of your data
* Defining the model you want to use
* Fitting of the model
* Use the model to make a prediction on some unseen data
* Scoring/evaluation of the accuracy of the model

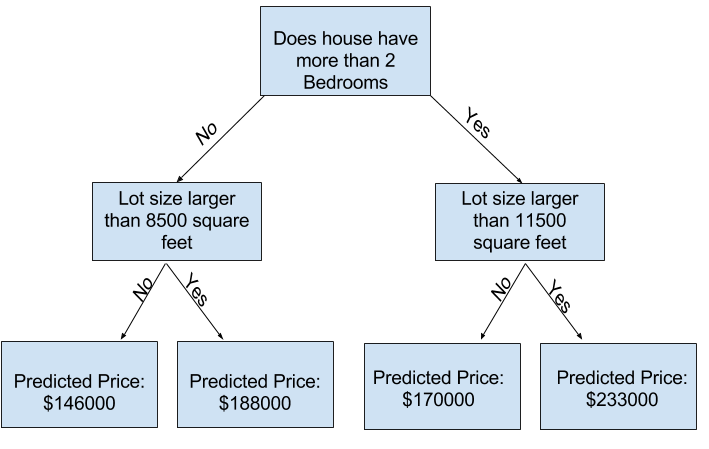
This example shows how to fit a regression model to predict the Total Work of Adhesion (TWA - A measure of stickiness) for some mucoadhesive polymers based on the same dataset we have been looking at previously.

The model we will be fitting to predict is called a “decision tree”. This is described in detail at the Kaggle course (<https://www.kaggle.com/code/dansbecker/how-models-work> ). In simple terms, we can make splits in the data to start predicting our label. For example, if we are trying to predict house prices, the below may be true:



This is obviously quite arbitrary, but in the process of **fitting**, the model will test different cutoffs (2, 3, 4 etc. bedrooms) and come up with a split that gives the most accurate result.

We can build up deeper and more complex trees to try and include the effects of multiple variables:

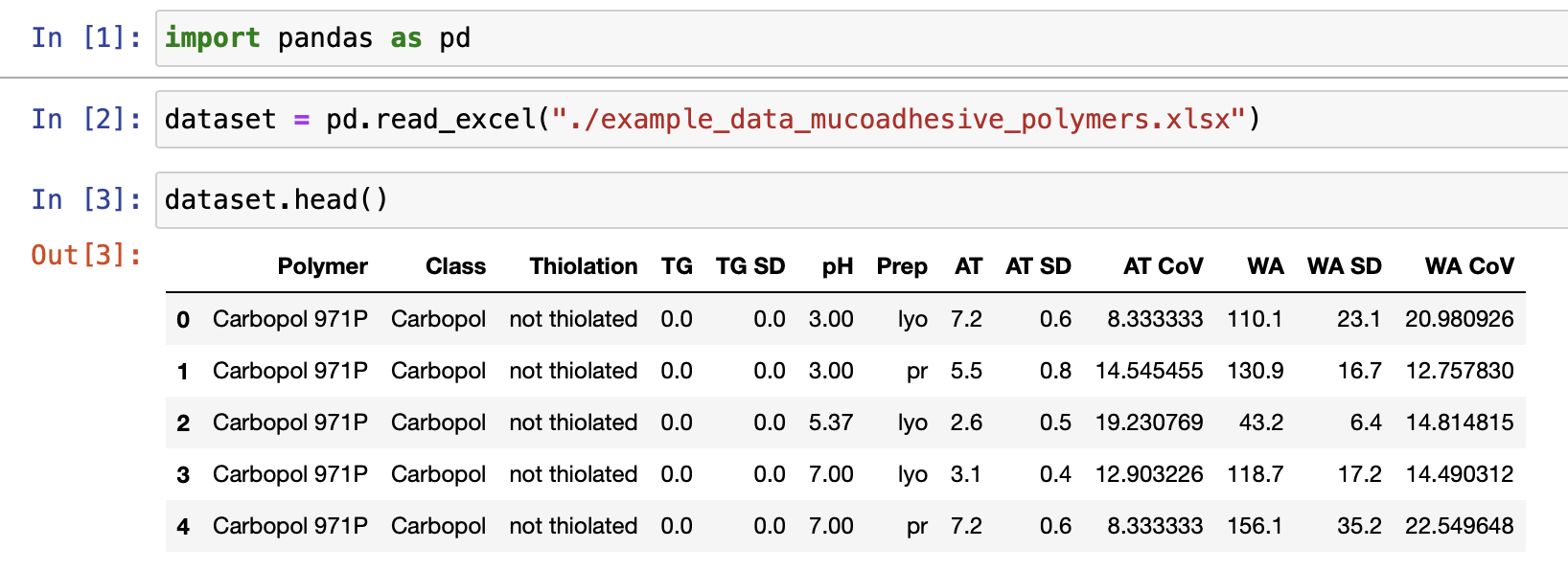


These trees can get extremely complicated, and ultimately are one of the most accurate types of machine learning for many problems.

Lets build a decision forest to try and predict our polymer properties.

**Curation and cleaning of data:**

We first start by reading in our dataframe from excel similarly to before:



In this case, we have superfluous columns that we need to remove, and a couple of other preprocessing steps.

To start with we need to split our data into groups – the **features** that we think describe our data, and the **labels** that we want to predict. These are also often called the independent and dependent variables (because we think the labels depend on the features in some way). To convention in machine learning is to call the features **X** and the labels **y**.

A screenshot of a computer

Description automatically generated

Here we want to use information on the class of base polymer (“Class”), the number of Thiol groups (“TG”), the pH of the solution (“pH”), the preparation method (“Prep”), and the Adhesion Time (“AT”) that the polymer stuck to a wheel to predict the Total Work of Adhesion (“WA”).

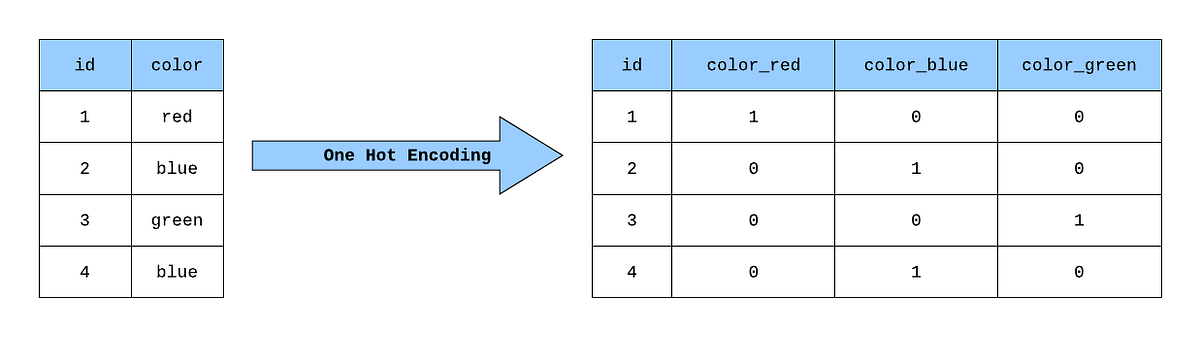
In practice this should be fairly easy because AT is a characterisation of the material similar to WA, and both should be highly correlated. The real challenge is to do this without knowing AT for the polymers as well.

**One-hot encoding**

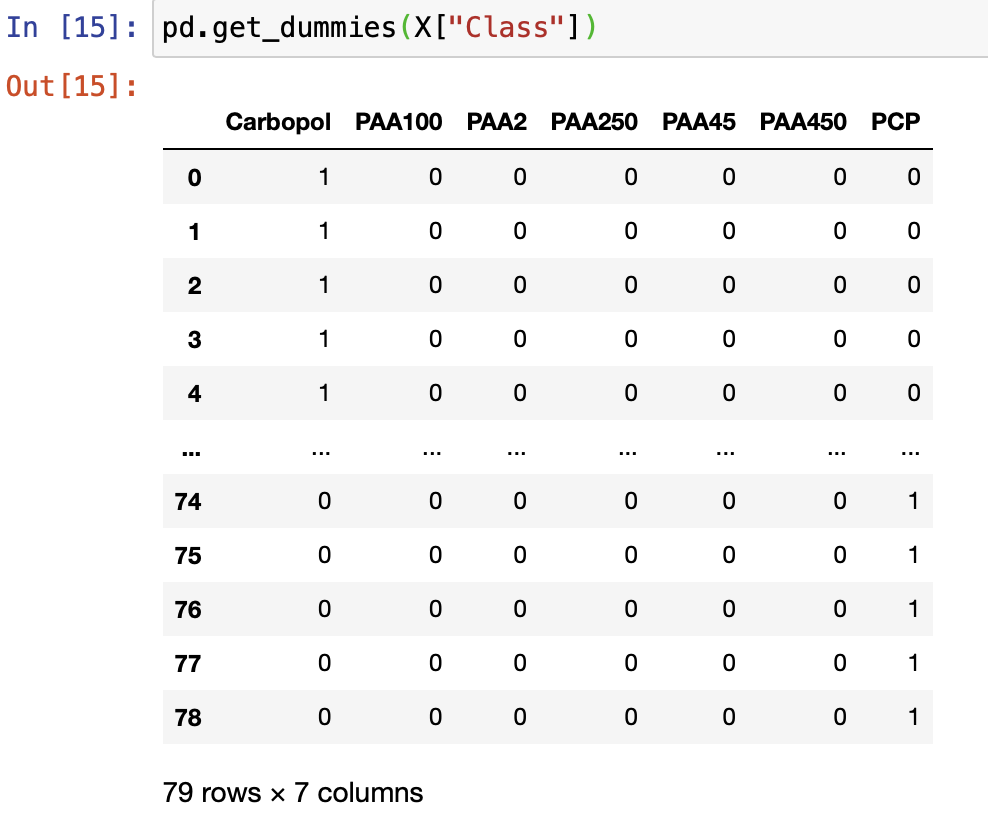
One quirk of machine learning is that inputs have to be some kind of numeric value. Some of our data fits this metric, but two columns, “Class”, and “Prep” are what are called **categorical**, meaning that they consist of different categories. One easy solution would be to convert the column to integers, for example, the “Class” column could just become a series of numbers from 1 to N. This is a problem in practice however – as machine learning algorithms read into the order and values of these categories, and will see a “Class” of 2 as being of greater value than 1.

Longer explanation here: <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

The way we solve this is to perform “One-hot encoding”. Essentially, we convert the categorical column to a series of binary columns representing the presence or absence of a class:

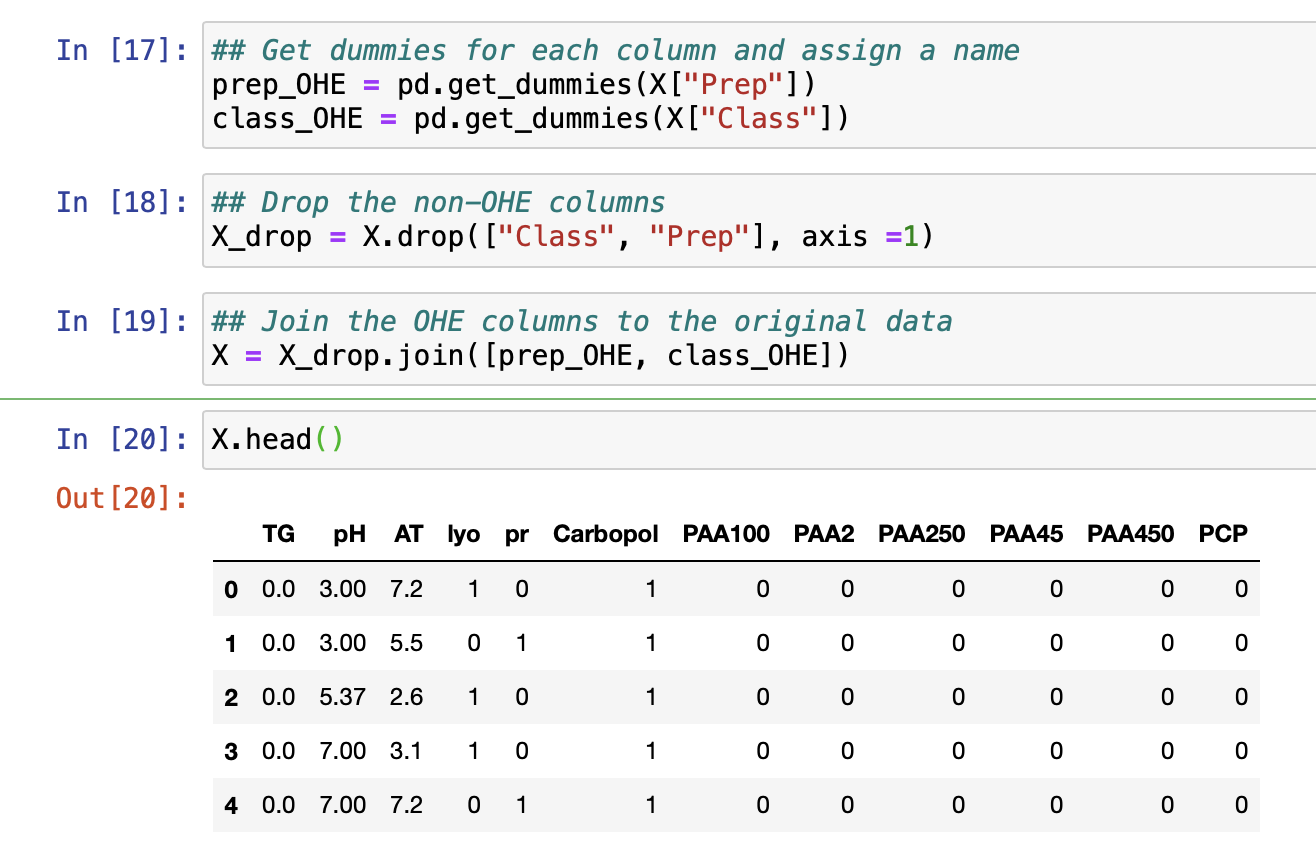


Luckily this can be performed automatically using the **“get\_dummies”** command in pandas.

A screenshot of a cell phone

Description automatically generated

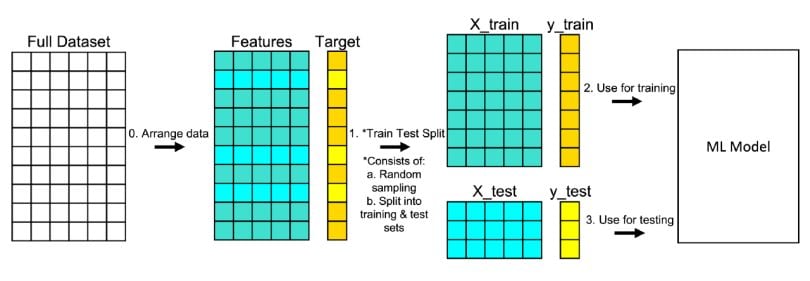
Goes to:

We also need to do a little pandas manipulation to replace the original column in the dataframe:

This falls into the loose category of work known as “data engineering”, and there are many ways of manipulating data to make it more amenable to performing ML on it.

**Splitting data**

In order to test how good our model is, we next need to split our data into a training set and a testing set. This is because the only way to know how good a model truly is, is to apply it to some data its never seen before and measure its accuracy. The best way to do this is to randomly sample some of your original data and then **hold it back** from the training of the algorithm. Once weve fit our model to the **training** data, we then apply it to the **test** data to see how good it is.



The library we will use to perform ML – **sklearn**. Can do this for us:

A screenshot of a computer

Description automatically generated

Here we pass the “train\_test\_split” function our X, y, the proportion of the split we want, and give it a random number (for reproducibility). Typically the split is performed on between 20% and 40% for the data, but this will depend on how much data you have.

**Defining the model**

We next need to define the model we are going to use. For now we will use a decision forest, as they are easy to understand. In practice you generally need to test many models to find the best, as there is no universally best ML method. This is known as the “no free lunch theory”: <https://machinelearningmastery.com/no-free-lunch-theorem-for-machine-learning/>

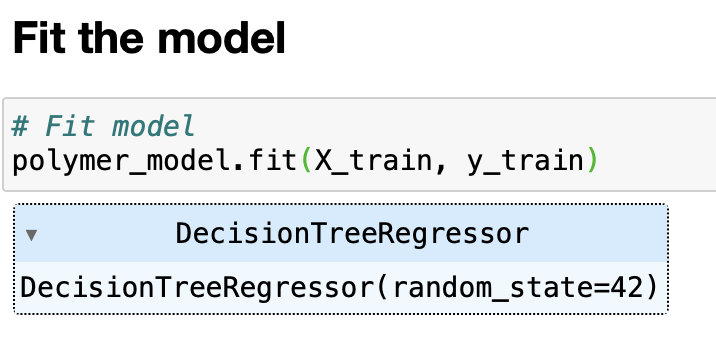
Here we define a decision tree regressor, with the default parameters:

A screenshot of a computer code

Description automatically generated

**Fitting the model**

Fitting the model consists of feeding the training features (X\_train) and the training labels (y\_train) to the decision tree, and asking it to learn how y depends on X. This is extremely simple:

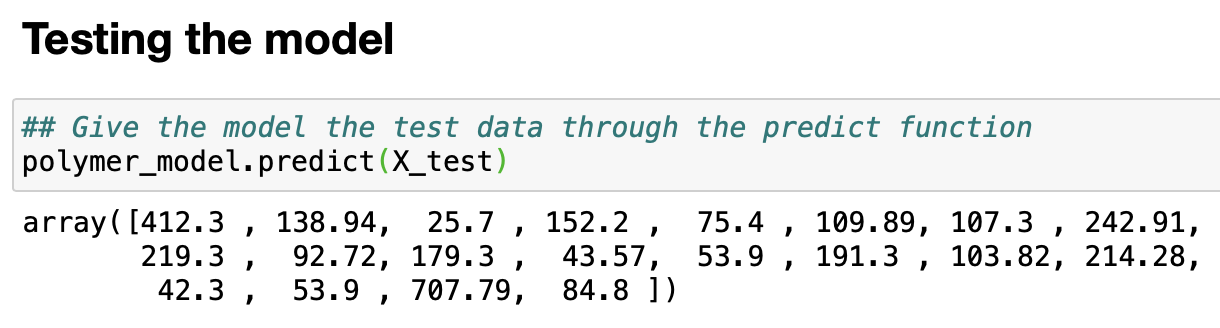


We now have our fit model – simple. (Whether its any good is a different story)

The best way to test our model is to apply it to the data we held out (the testing data) and see how well it performs:

**Testing the model**

We can apply the model to new data extremely easily using the “predict” function. If we just pass the model our testing data through this function it will tell us what it thinks the y value is for each member of our test set.



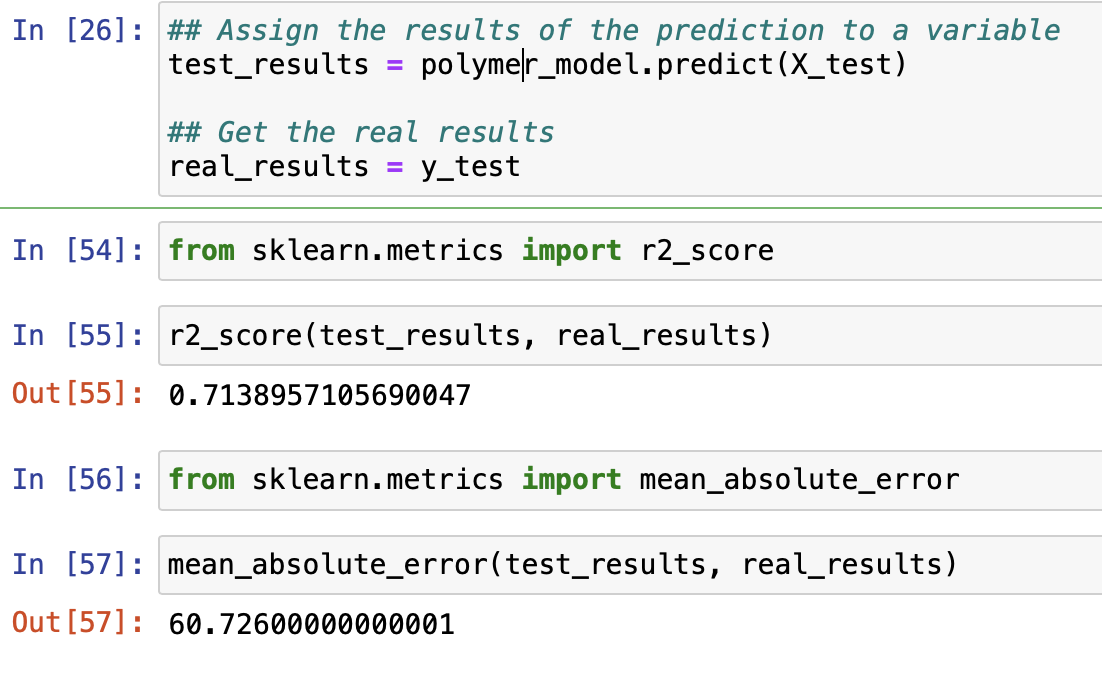
We can now compare this to the real values for the these samples. There are many ways we can “score” a model. Here is a discussion on some of them:

<https://towardsdatascience.com/ways-to-evaluate-regression-models-77a3ff45ba70>

In this case, we will use two methods:

* R squared (r2). This is a measure of how much of the variance in y is learned by the model. Essentially a value of 1 is perfect, and 0 is if you just predicted the mean of y for every sample. This value can be negative, meaning a model is so poor it is worse than the mean. Negative r2 means a crap model and probably overfitting.
* Mean absolute error (MAE). This is a measure of how far off we are from the “true” value on average. Essentially we take the difference between the predicted and real values, make them all positive (the absolute value), and average them. The result is a measure of how much we are off in our prediction on average. Bare in mind – this value is dependent on the range of the data. An MAE of 1 is great if the data ranges between 0 and 1000, but not great if it ranges from 0 to 1.

Getting these scores is easy in sklearn:

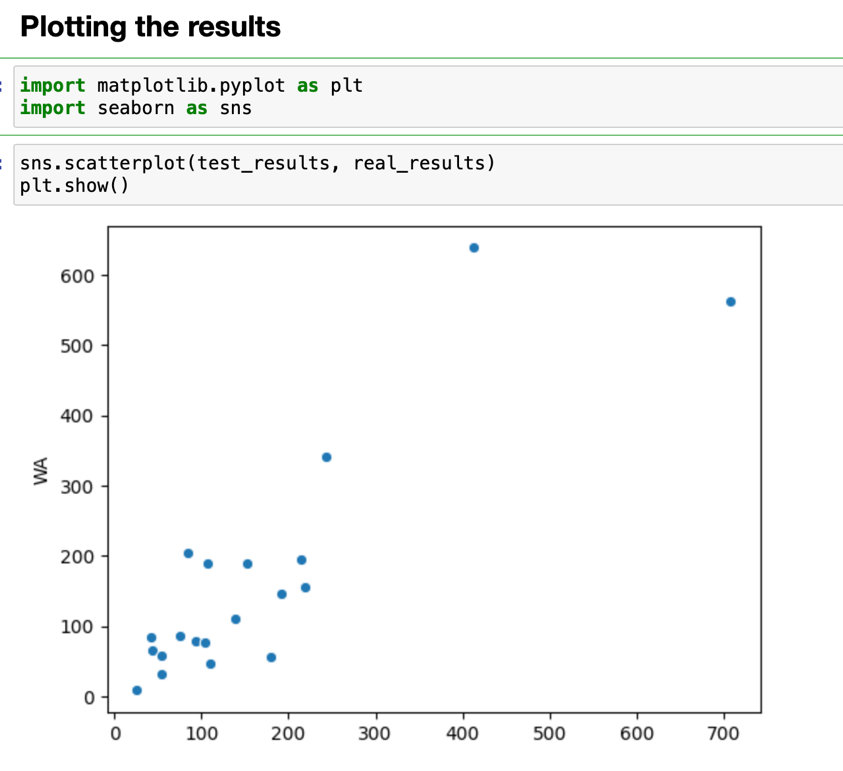


We have and r2 of 0.71, and an MAE of 60, which is not bad (but not unexpected since we are passing the model Adhesion Time (AT) as a feature.

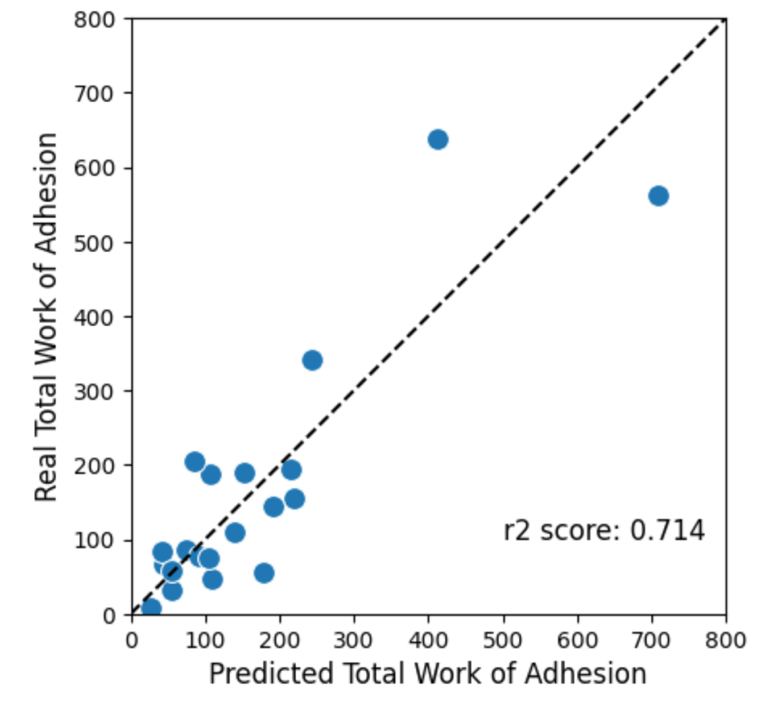
Finally, we can plot the results.

**Plotting the results**

Using our knowledge from the previous workbooks:



And spending a little time making this look nicer:



Not a bad start.

This has been a very simple introduction to supervised machine learning. The next things we need to consider are:

* We only used the default decision forest. There are many things we can change about the algorithm to make it better (optimisation).
* We chose a random 25% of the data to “hold out”. How dependent is our model on which 25% was chosen? There are ways we can start to unpick this (cross validation).

A next step will be to look at Kaggle: <https://www.kaggle.com/learn/intermediate-machine-learning>