# Data Analysis & Machine Learning

Lecture 2:

***Data Science Tools & Decision Trees***

Robert Currie

## Intro

Who am I?
Physicist turned Computing Researcher/Admin/Developer/…

Did my PhD on LHCb experiment at Edinburgh in 2009-2014.

What will I be covering today?

- Intro to (some of) the software used to perform ML
- Quick run-through of some useful features
- Intro to Decision Trees
- How best to fit Decision Trees to data

# ML/AI Software

- Many software libraries support ML style workflows.

  I will be exclusively using Python and Jupyter notebooks for this course.

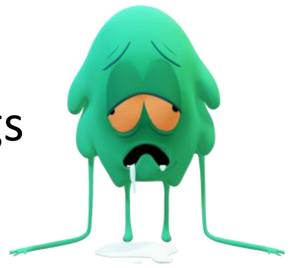
- Advantages:

- simpler to develop for, very dynamic
- easy to learn
- can express complex problems using simple code
- wide adoption of language in industry

- once setup, need no code changes* to access GPU, TPU, FPGA…

- Disadvantages:

- potential performance overhead(s)
- abstract interface is far away from problems/bugs
- installation/setup can be… tricky…



*well, almost no code changes…

# ML/AI Software – Frontend(ish) Projects

Typically speaking tend to use *just 1 or 2* of these per project/problem.

- **Jupyter** *(notebook, lab, …)*

  Biggest piece of software you will likely directly encounter doing data analysis/ML.

- **TensorFlow**

  This is the biggest library, used for building ML models for training, predicting & generating data.
  Uses Tensor objects to pass data around.

- **Keras**

  This can be thought of as the model component within TF but is developed independently.
  Used for generating Neural Net models.

- **SciKit Learn**

  This project is widely used for classification, regression and clustering style problems.

- **PyTorch**

  This is used for building large AI models and is widely used in production by industry.

Icons from Wikipedia/project home-pages

# ML/AI Software – Backend(ish) Projects

Supporting projects, used as needed.

- **Numpy**
  This framework is a widely used Python numerical framework
- **Pandas**
  Pandas is a library built around making an easily manipulated data frame
- **Matplotlib**
  This library is used for visualising data that is generated in Python and builds atop the Numpy framework
- **Seaborn**
  Statistical visualization framework built atop Matplotlib
- **SciPy**
  This framework is built atop numpy and provides code to help with high-level mathematical functions such as fft, linear algebra, integration…

Icons from Wikipedia/project home-pages

For Reference

## ML/AI Software – Notable other tools

- **Python virtualenv**
  Most common and widely used tool for building environments to install Python packages as/when needed as a user.
- **Conda**
  A system which is like VirtualEnv but distributes binary and Python software which can be managed by a user. Also offers cross-platform common environments.
- **Plotly**
  This is a data visualization framework which is gaining growing adoption
- **Theano**
  Framework used for doing tasks like Matrix manipulation using CPU and accelerators
- **Caffe**
  This library is the PyTorch model library which is like Keras for TensorFlow
- **Git**
  This is the most common version control system for tracking changes in documents

## ML/AI Software - *A word of caution*!

- It’s 2023, Python2 is now long dead!
  *(If you’re too young to remember it’s ok)*

- Tensorflow2 is very different to Tensorflow1
  *(beware old examples online, TF2 is easier/better and it’s now more pythonic)*

- iPython is basically Python3 with extras, but there are *minor* differences…

- Save regularly and don’t be afraid of trying/breaking things ☺

# ML/AI Software HowTo

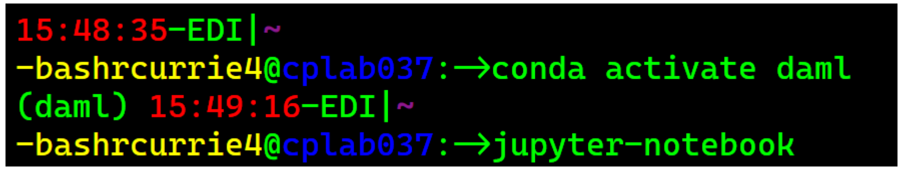
- Labs machines are connected to via: https://www.wiki.ed.ac.uk/display/pastudentinfo/School+Student+Computing+Facilities

- Using Anaconda in labs: https://www.wiki.ed.ac.uk/display/pastudentinfo/Using+Anaconda+and+Spyder+on+CPLab+PCs

# ML/AI Software HowTo

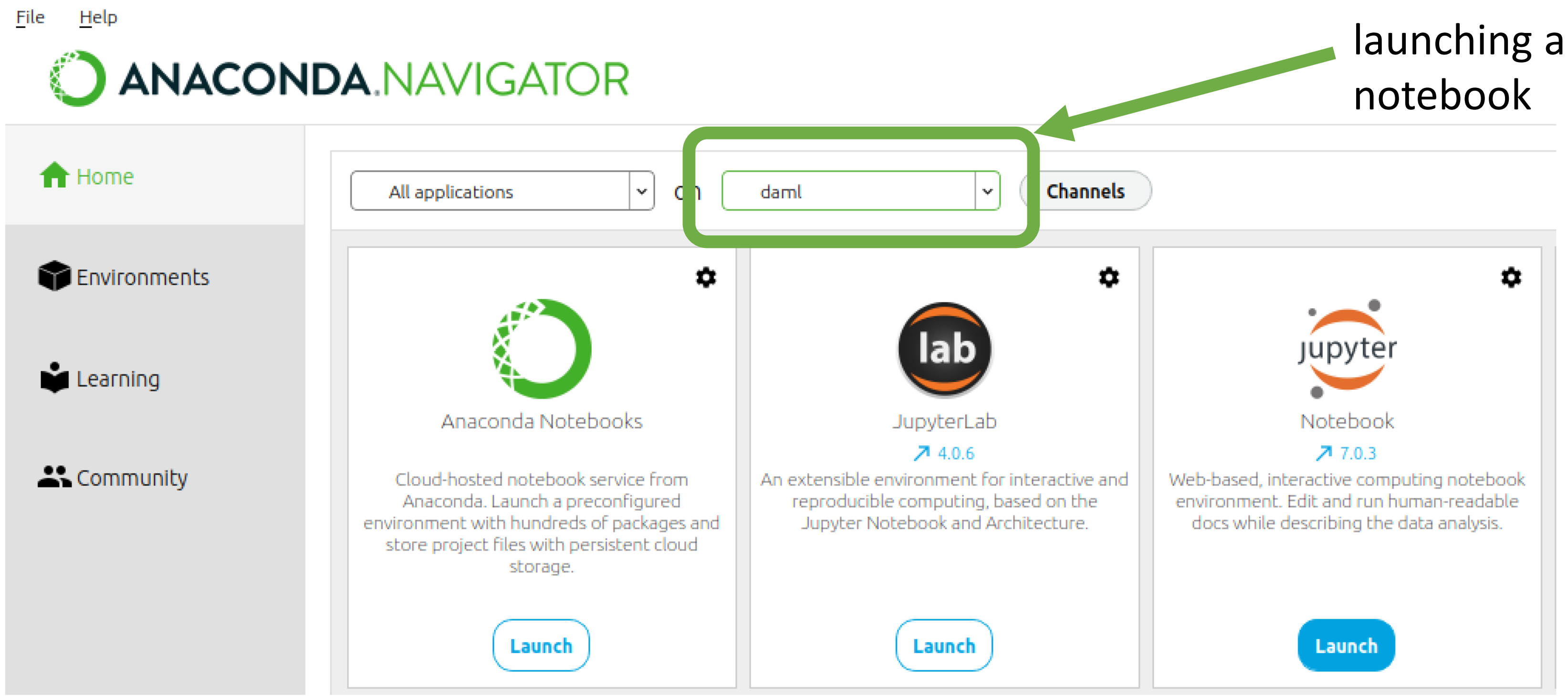
Make sure that you launch Jupyter notebooks from within the correct Anaconda environment:

Terminal:



```
15:48:35-EDI|~
-bashrcurrie4@cplab037:→conda activate daml
(daml) 15:49:16-EDI|~
-bashrcurrie4@cplab037:→jupyter-notebook
```

GUI:



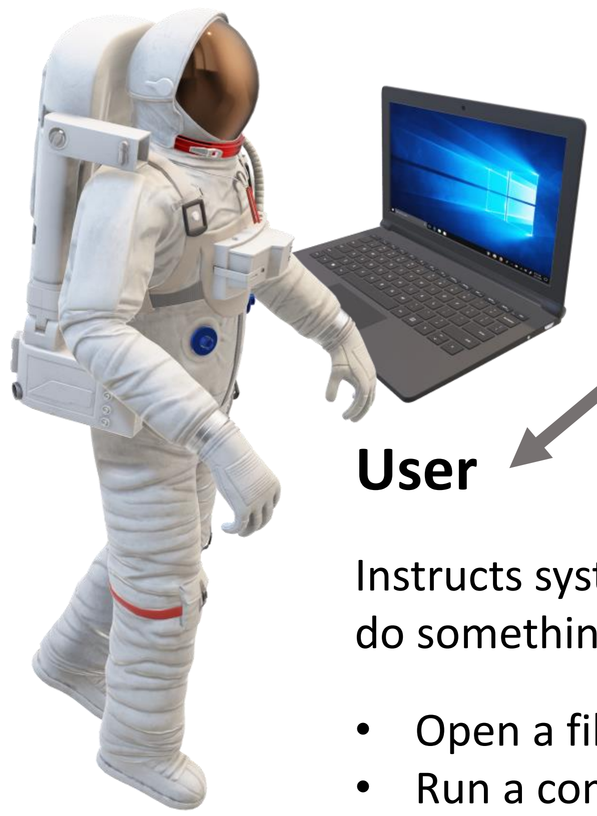
Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Select the daml environment **BEFORE** launching a jupyter notebook

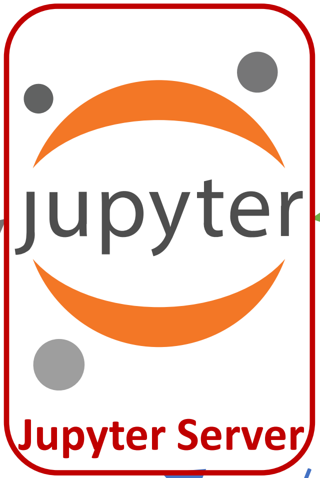
# ML/AI Software – Jupyter notebook



**User**

Instructs system to do something e.g.:

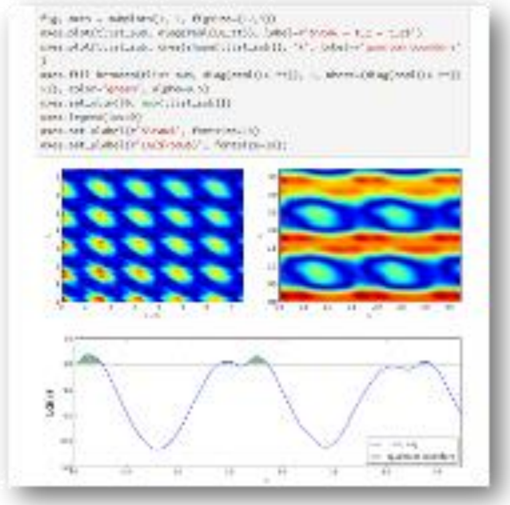
- Open a file
- Run a command
- Draw an image
- …



**Kernel**

Where the work/processing is done

Support for different languages: Python, Julia, Spark, R, …

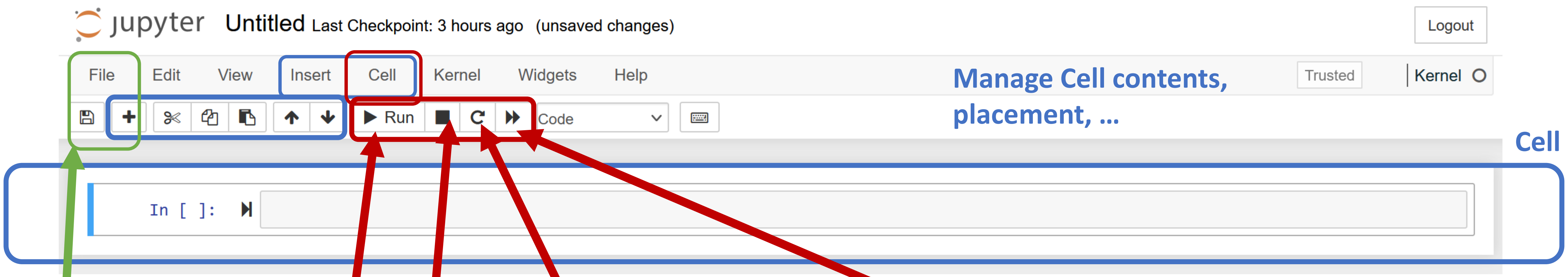


**Jupyter Notebook file**

Used for saving state, output, …

**someFile.ipynb**

# ML/AI Software – Jupyter notebook


jupyter
Untitled Last Checkpoint: 3 hours ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Kernel
Run
Code
In [ ]:
Manage Cell contents, placement, …
Cell

File operations, open/close/save/…

Run the selected cell(only) now

Stop the current execution.

(Same as Ctrl+C or killing the process via the host-OS, technically interrupts the notebook kernel)

Restart the Kernel

Beware this deletes all transient variables.

Reset the Kernel and re-run all the cells in the notebook

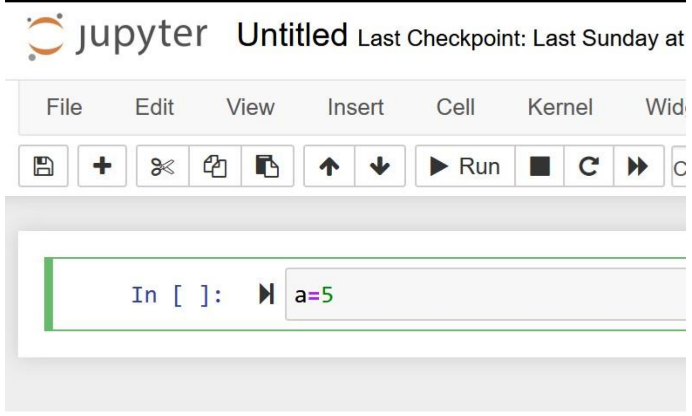
(will stop on error/exception)

*My personal advice:*

*This is expensive/slow but it's worth repeating regularly(!)*

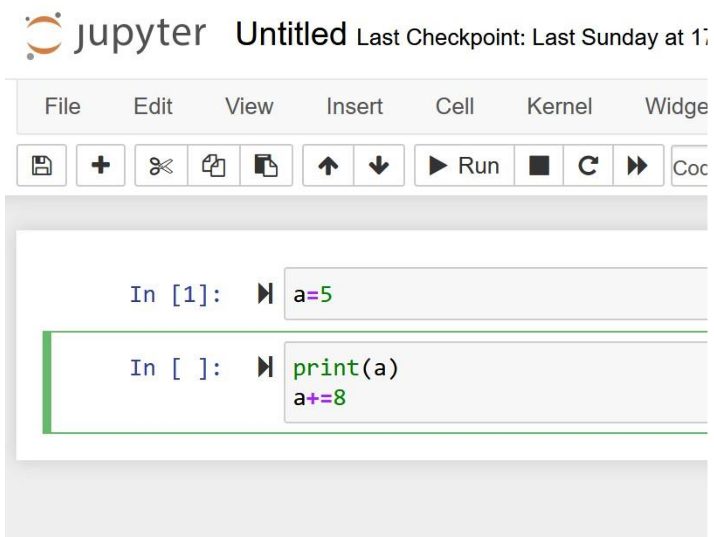
# ML/AI Software – Jupyter notebook

**1)** First type command in the cell...



```
In [ ]: a=5
```

**2)** Shit+Enter executes cell command(s)

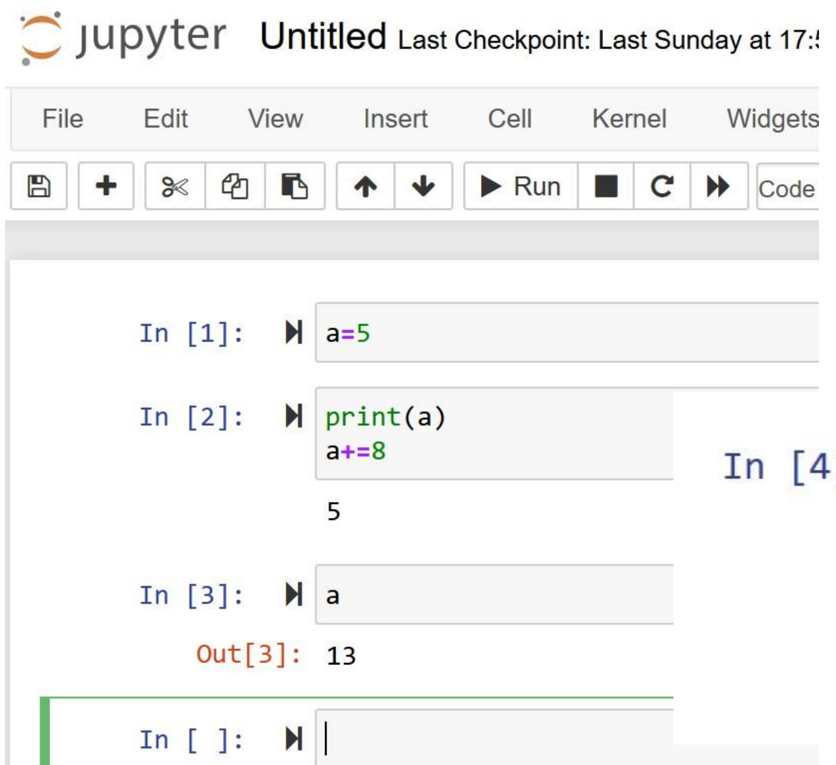


```
In [1]: a=5

In [ ]: print(a)
        a+=8
```

**3)** Using return within a cell allows for multiple commands per-cell.

**4)** Output from print(a) command gets saved.

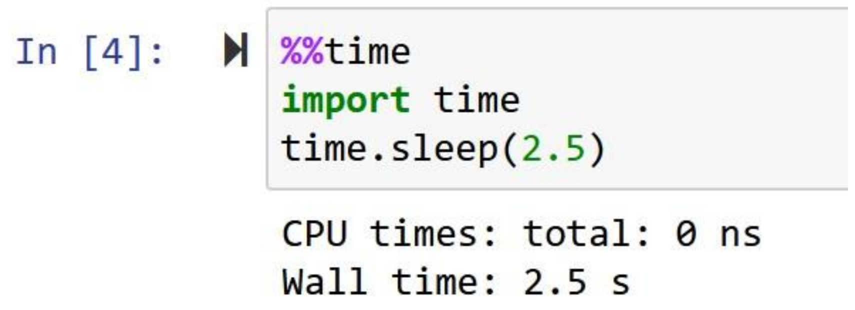
**5)** Executing an object prints the string value as in interactive Python.



```
In [1]: a=5

In [2]: print(a)
        a+=8

        5

In [3]: a

Out[3]: 13

In [ ]: |
```

6) **%%magic** commands at top of cells have special behaviour!

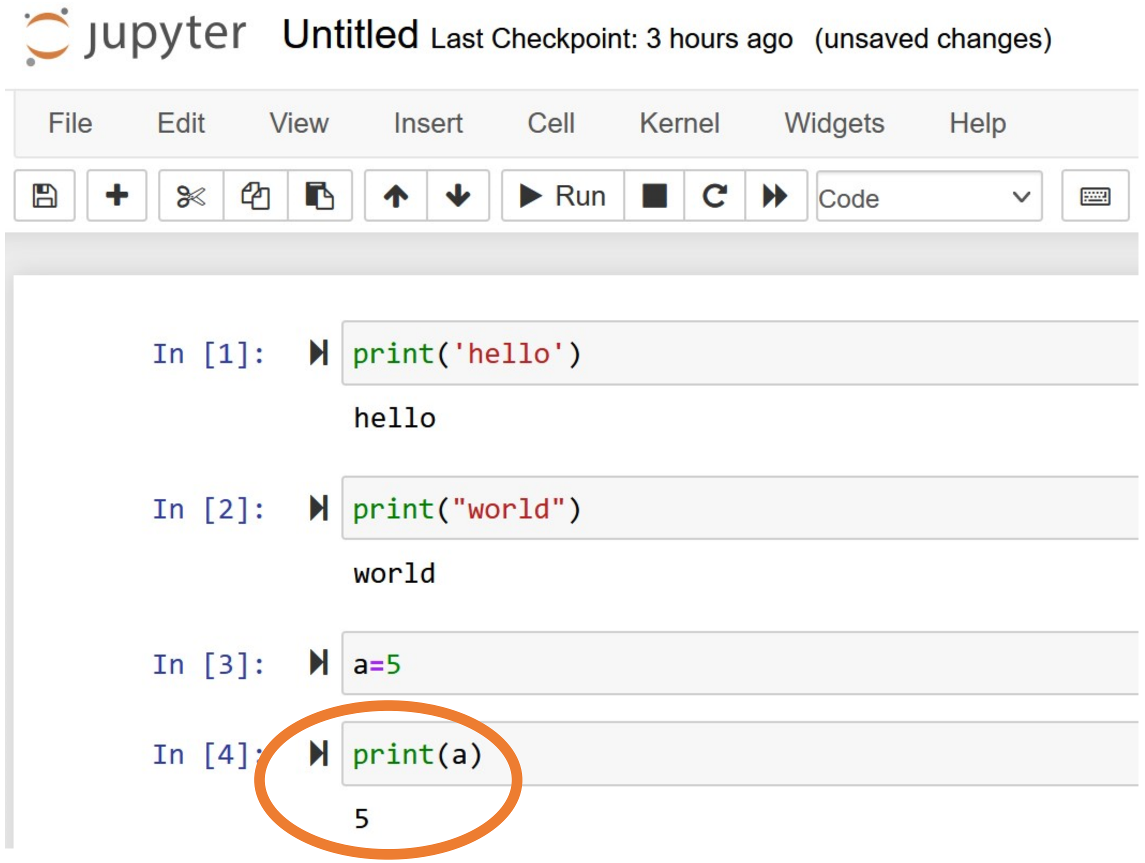
**%%time** allows us to time the execution of a command



```
In [4]: %%time
        import time
        time.sleep(2.5)

        CPU times: total: 0 ns
        Wall time: 2.5 s
```

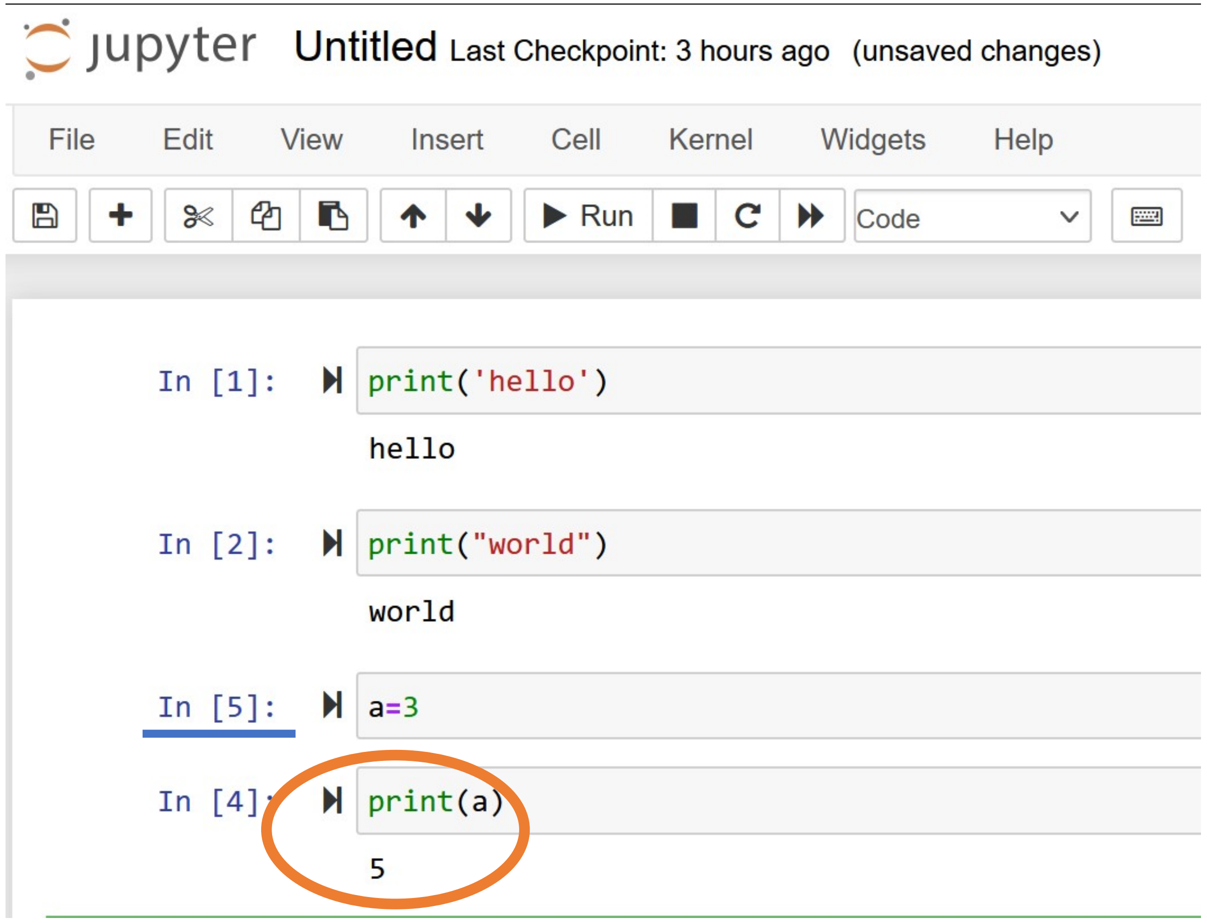
# ML/AI Software – Jupyter notebook

## Good + makes sense


jupyter
Untitled Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [1]:  print('hello')
         hello

In [2]:  print("world")
         world

In [3]:  a=5

In [4]:  print(a)
         5
```

## Bad + Confusing(!)

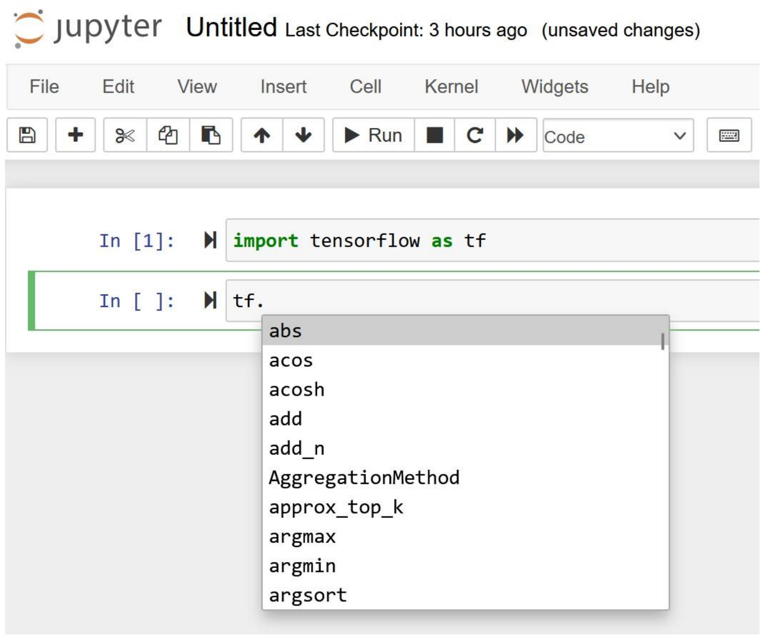

jupyter
Untitled Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [1]:  print('hello')
         hello

In [2]:  print("world")
         world

In [5]:  a=3

In [4]:  print(a)
         5
```

# ML/AI Software – *using* Jupyter notebooks

**TAB complete <*TAB*>** i.e.

hitting tab after the `.` to see object attributes/functions in a scrollable menu

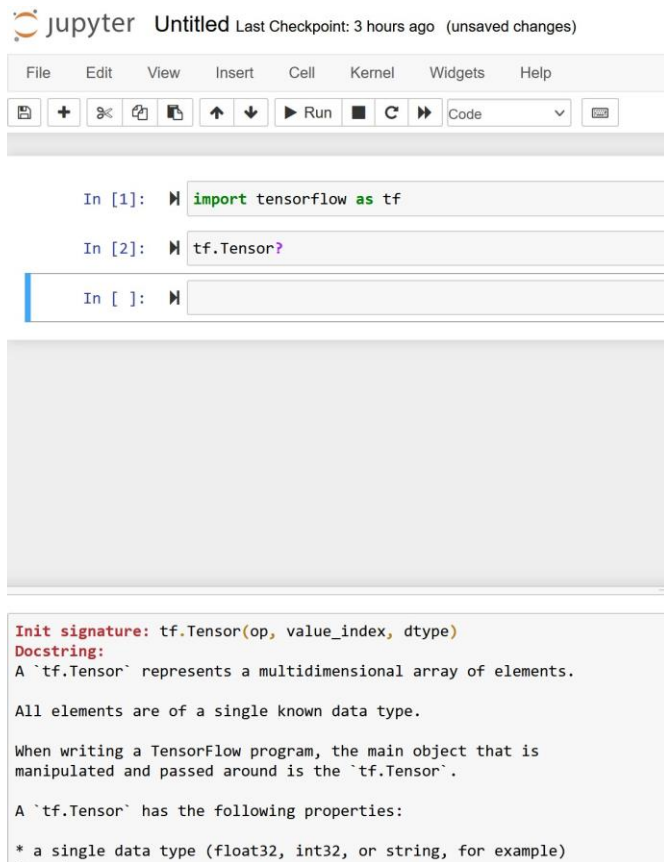

jupyter Untitled Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run Code

```
In [1]:  import tensorflow as tf
In [ ]:  tf.
```

```
abs
acos
acosh
add
add_n
AggregationMethod
approx_top_k
argmax
argmin
argsort
```

**Documentation quick-access: ?**

Execute `object?` to see the docs on the object

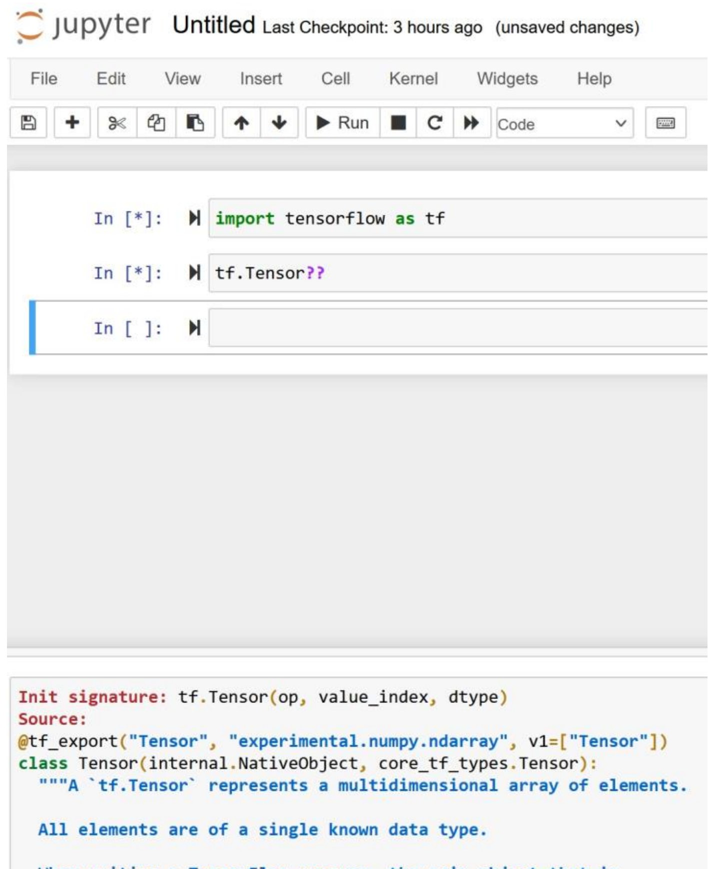

jupyter Untitled Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run Code

```
In [1]:  import tensorflow as tf
In [2]:  tf.Tensor?
In [ ]:
```

```
Init signature: tf.Tensor(op, value_index, dtype)
Docstring:
A `tf.Tensor` represents a multidimensional array of elements.

All elements are of a single known data type.

When writing a TensorFlow program, the main object that is
manipulated and passed around is the `tf.Tensor`.

A `tf.Tensor` has the following properties:

* a single data type (float32, int32, or string, for example)
```

**Source view: ??**

Execute `object??` to see the source of the object


jupyter Untitled Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run Code

```
In [*]:  import tensorflow as tf
In [*]:  tf.Tensor??
In [ ]:
```

```
Init signature: tf.Tensor(op, value_index, dtype)
Source:
@tf_export("Tensor", "experimental.numpy.ndarray", v1=["Tensor"])
class Tensor(internal.NativeObject, core_tf_types.Tensor):
  """A `tf.Tensor` represents a multidimensional array of elements.

  All elements are of a single known data type.
```

## ML/AI Software – *using* Jupyter notebooks

Jupyter notebooks tie into external libraries to collect (and save) the output into the notebook.

Notebooks do NOT save the transient(running) state of the kernel.

This means you can't expect to open a notebook and re-run "cell 56" without first running cells 1-55.

# ML/AI Software – *using* Jupyter notebooks

There's an example notebook: "**data-science-tools.ipynb**" please read through it.

This notebook goes through examples raw data manipulation using:

- Numpy Arrays
- Statistics
- Pandas DataFrames
- Matplotlib

It's useful to read through this notebook, play with values, experiment with any notation you're not familiar with.

TensorFlow uses "Tensor"* object which behave *very similarly* to Pandas Dataframes.

*No relation to Tensors in Physics equations…

# Machine Learning – A brief History

1956

First Conference on AI research

Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI)

Robby the Robot in Forbidden Planet

1965

"Group method of data handling" First example of training a Deep Neural Network

1970s-1980s

AI winter, early promises were big, results small and computers slow…

1980s

Increase in funding and interest in research labs, Bell, IBM…

1994

MNist database developed

1997

Deep Blue beat Gary Kasparov a world chess champion

1998

LeNet-5 developed CNN shown to out-perform other models

2011/2012

AlexNet a CNN enters and wins multiple competitions on image recognition

2015

AlphaGo beats Fan Hui a professional go player

2021

Dall-E released showing possibility of of using large Diffusion models to generate images

2022

ChatGPT released. Large Language Model trained with hundreds of billions of parameters.

Lots of interest in how model is able to infer context from training on input.

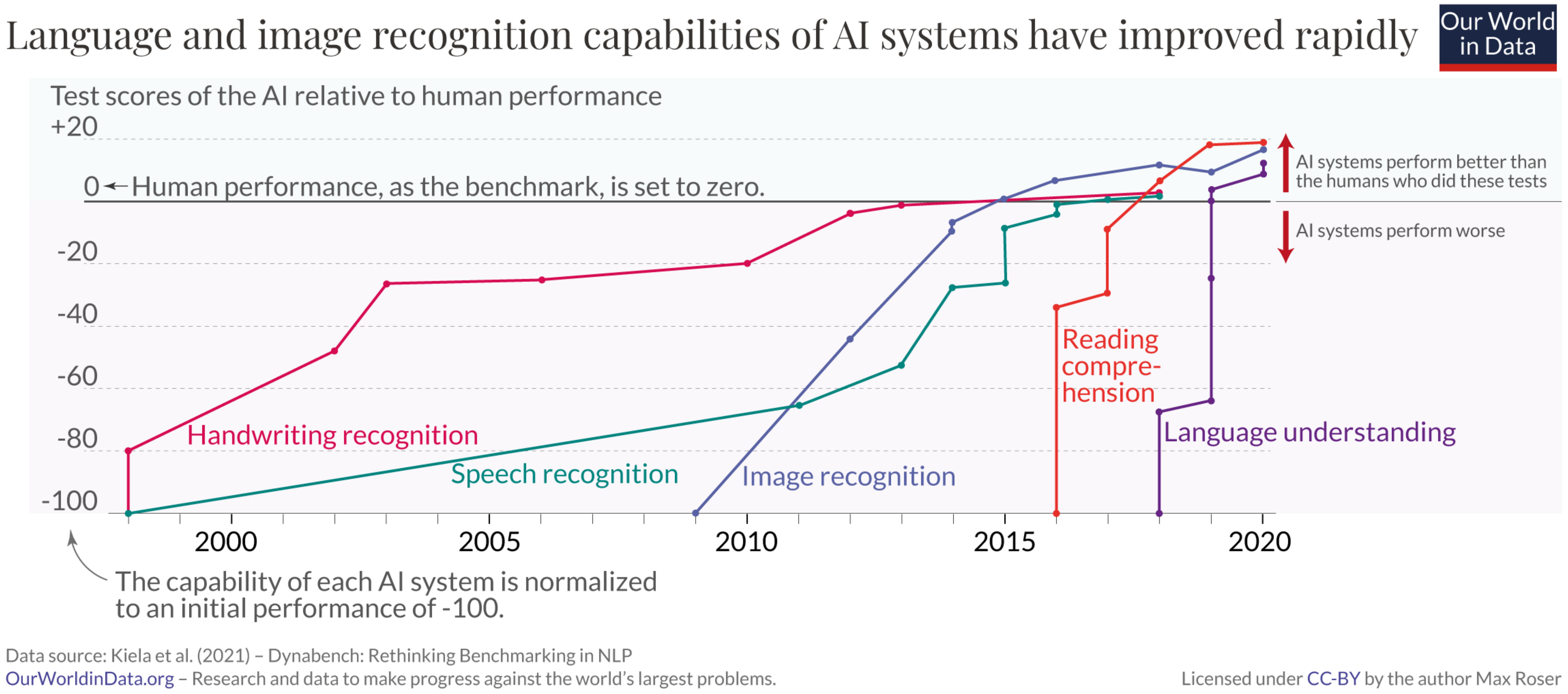
Output is effectively the best auto-complete ever invented.

?

???

"Teddy bears working on new AI research underwater with 1990s technology"

# Machine Learning – A brief History



## Machine Learning – Glossary

Terminology used when discussing fitting models in machine learning:

- **Model**
  This is the model which is being fitted, examples include Decision Trees, Deep Neural Networks and Generative adversarial networks.

- **Epoc**
  Number of steps in a fitting process that a model has been trained for.

- **Fit**
  The process where free parameters within a model are modified to more optimally describe the data.

- **Batch**
  The size of the data subset to be trained each time with a copy of the model.
  This is used when the dataset is so large that it can’t be fit on a single computing resource.

- **Iterations**
  Number of batches required to be processed within a single epoc.

## Machine Learning

Machine learning is a growing field. As Christos said last week, DAML is not intended to be a replacement for a course on AI/ML.

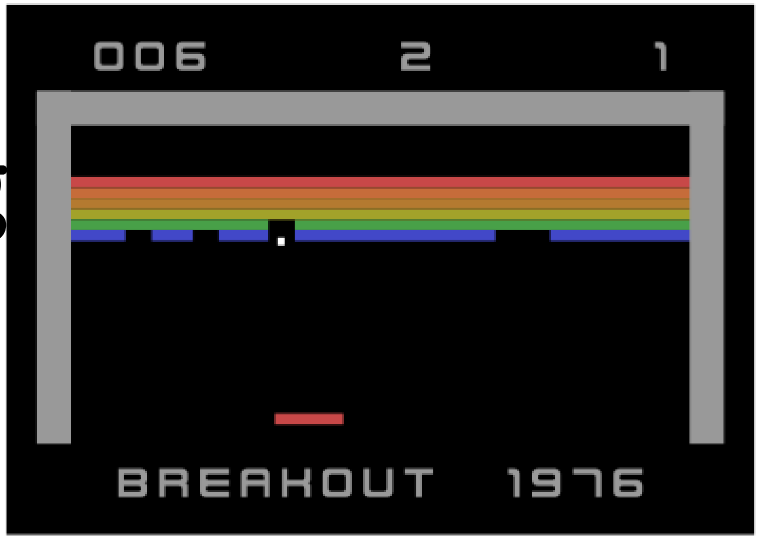
Generally, machine learning model types fit into 3 categories:

**Supervised**

**UnSupervised**

**Reinforcement**

# Machine Learning – Models/Fitting



- **Reinforced** learning models are models which are trained in an environment where there is a reward given to the model for certain actions.

- How to get the rewards is not intrinsically known to the model, and it's expected that the model will "discover the rules" required to achieve these.

- Easiest example in this case is something like a model to play a game, e.g. breakout, super-mario, etc…

  Also works for training AI to other game-ified systems with a score.

## Machine Learning – Models/Fitting

This Photo by Unknown Author is licensed under CC BY-SA

- **Unsupervised** and **semi-supervised** models are designed to run over large un-labelled datasets extracting features from the data itself.

- Intention is that the model is faster and/or better at extracting these relationships than an alternative hand-written/optimized model.

  Typically, generator models are **unsupervised** as they can be trained on much larger datasets to extract more data and encode more detailed relationships.

- **Semi-supervised** models often have a model trained on a smaller human labelled dataset that acts as a control to train a 2$^{nd}$ **unsupervised** model which runs over a much larger dataset with more free parameters.

- The difference between **unsupervised** and **semi-supervised** can often get blurred.

# Machine Learning – Models/Fitting

- **Supervised** models are generally split into Regression and Classification. Used in fitting data where the output is known.

  This means dataset output has been labelled, e.g. simulated data, labelled photographs, data classified by hand…

- **Supervised** datasets have a larger training set and a smaller testing set.

  Trained models are tested against the testing dataset to assess performance and over-training.

- Common types of **Supervised** models include Decision Trees and Neural Networks as well as Support Vector, Logistic/Linear, …

  Now let's discuss Decision Trees.

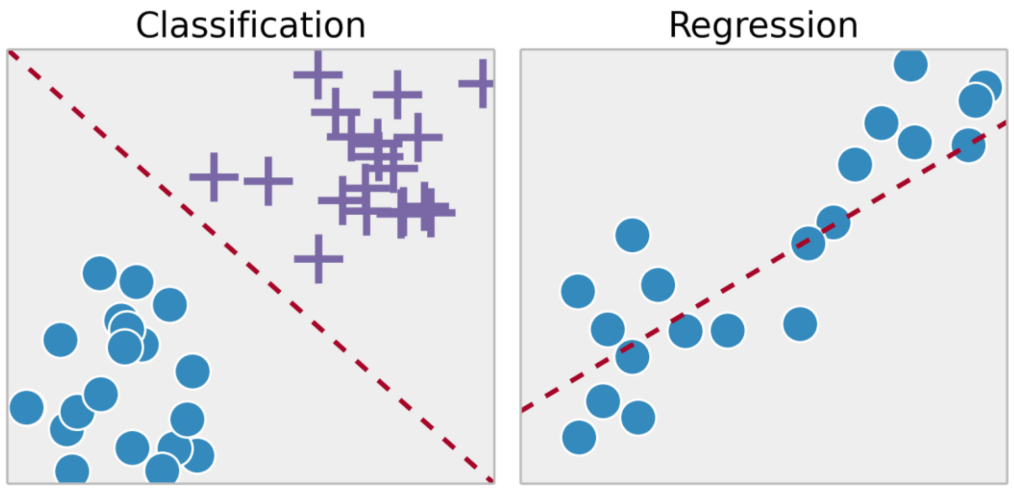
# Decision Trees Intro

Decision trees are among the conceptually simplest of ML models.

This type of machine learning model is well suited to problems such as classification and regression.

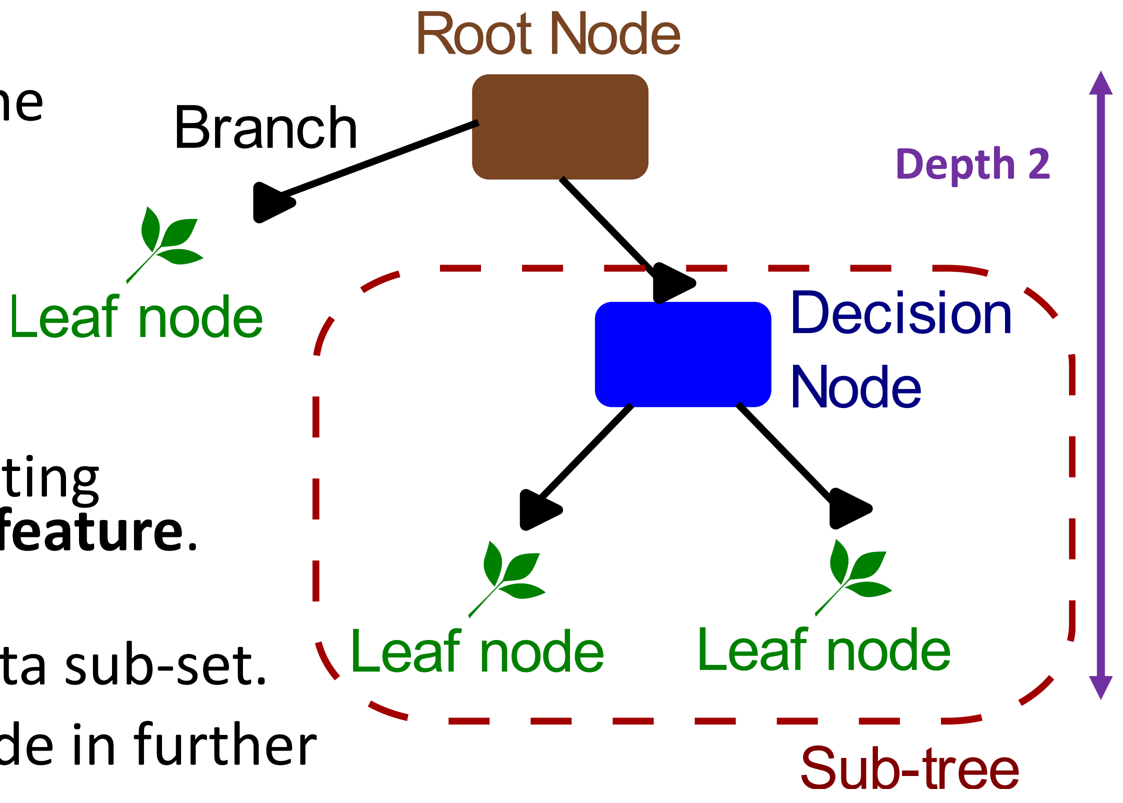
A ***classification*** model can predict whether a datapoint is a certain ***class***.

A **regression** model can predict a ***trend based*** on continuous data.



# Decision Trees – The Anatomy of

- The **Root node** of a decision tree is where the first selection is made.
- Decision Trees are formed from multiple nodes connected by branches.
- Nodes are where a selection is made separating the dataset based on a decision based on a **feature**.
- A **Leaf node** ideally would contain a pure data sub-set.
- Further **decisions** based on features are made in further nodes within a tree.
- Many implementations of Decision Trees limit each decision to 2 outcomes. In this case, the tree can be referred to as a "Binary Decision Tree".



# Decision Trees – How to Decide?

- Want to select Green from Orange

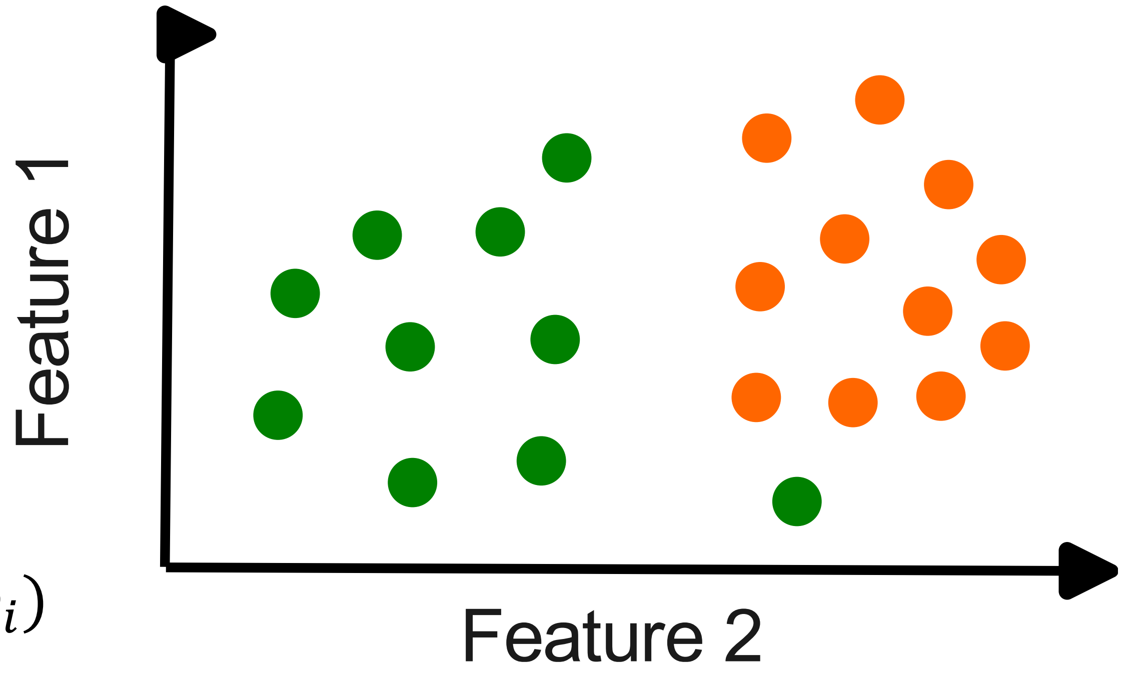
- Best decision is made to:

  - Minimize Entropy: $E = \sum_{i=1}^{n} -p_i \log(p_i)$

  or,

  - Minimize Gini index: $Gini = 1 - \sum_{i=1}^{n} (p_i)^2$

  *Also known as "Gini impurity" or "Gini diversity" index*



- Other decision such as maximizing 'information gain' exist. For now, we'll consider only Entropy and Gini index

# Decision Trees – Making a Decision

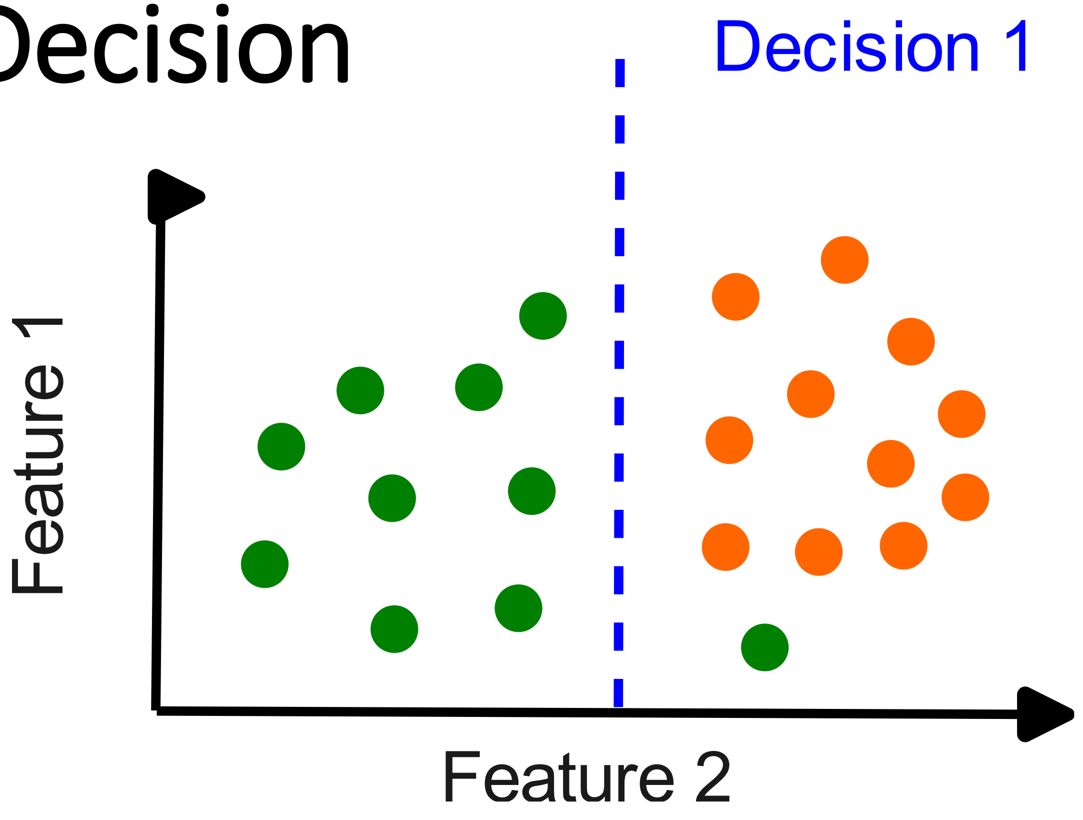
- Root Node makes the decision which leads to the lowest Gini score.

- Does Decision 1 give us a good Gini value?

$$Gini(right) = 1 - \left(\frac{1}{11}\right)^2 - \left(\frac{10}{11}\right)^2$$
$$= 1 - 0.008 - 0.826$$
$$= 0.166$$

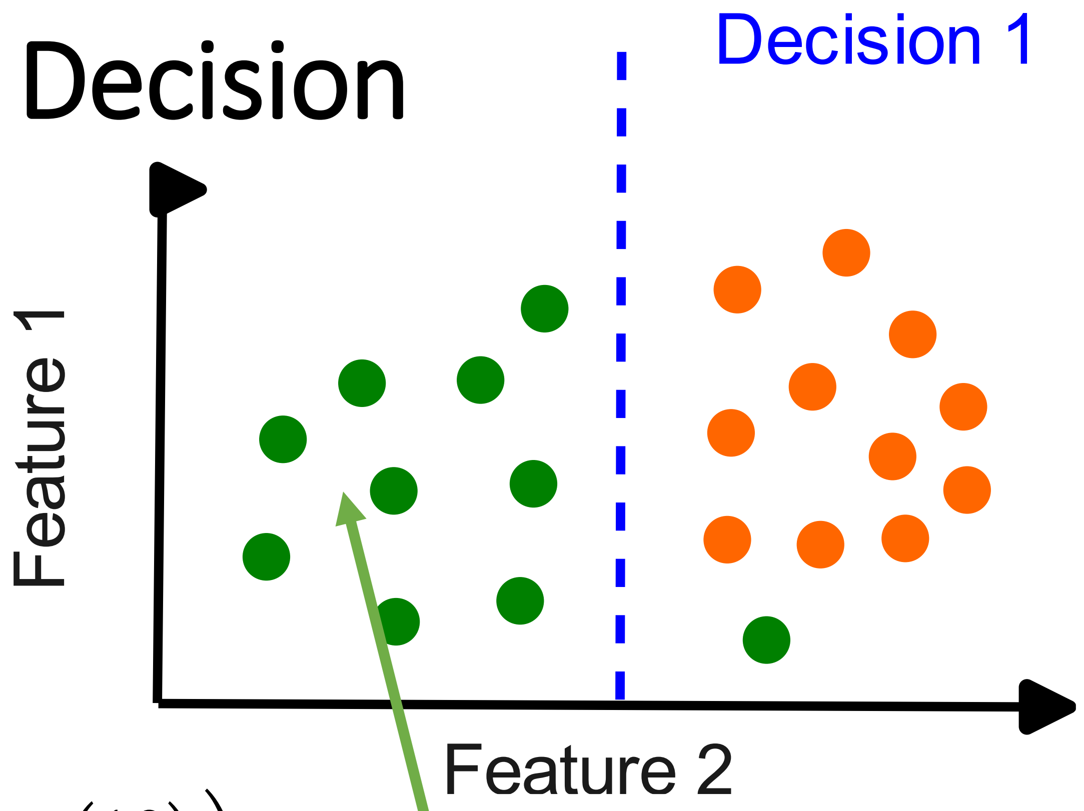
$$Gini(left) = 1 - 1^2 = 0$$

$$Gini(average) = 0 \times \frac{9}{20} + 0.166 \times \frac{11}{20} = 0.09$$



# Decision Trees – Making a Decision

- OK, we decided to select first on "Feature 2".
- Does that give us a good Entropy value?



$$E = -\left(\mathbf{1 \times \log(1)} + \frac{1}{11} \times \log\left(\frac{1}{11}\right) + \frac{10}{11} \times \log\left(\frac{10}{11}\right)\right)$$
$$= -(\ \mathbf{0}\ +\ -0.09\ +\ -0.038\ )$$
$$= 0.13$$

Pure sub-sample of species 1, we can call this a Leaf node. No further decisions need to be made.

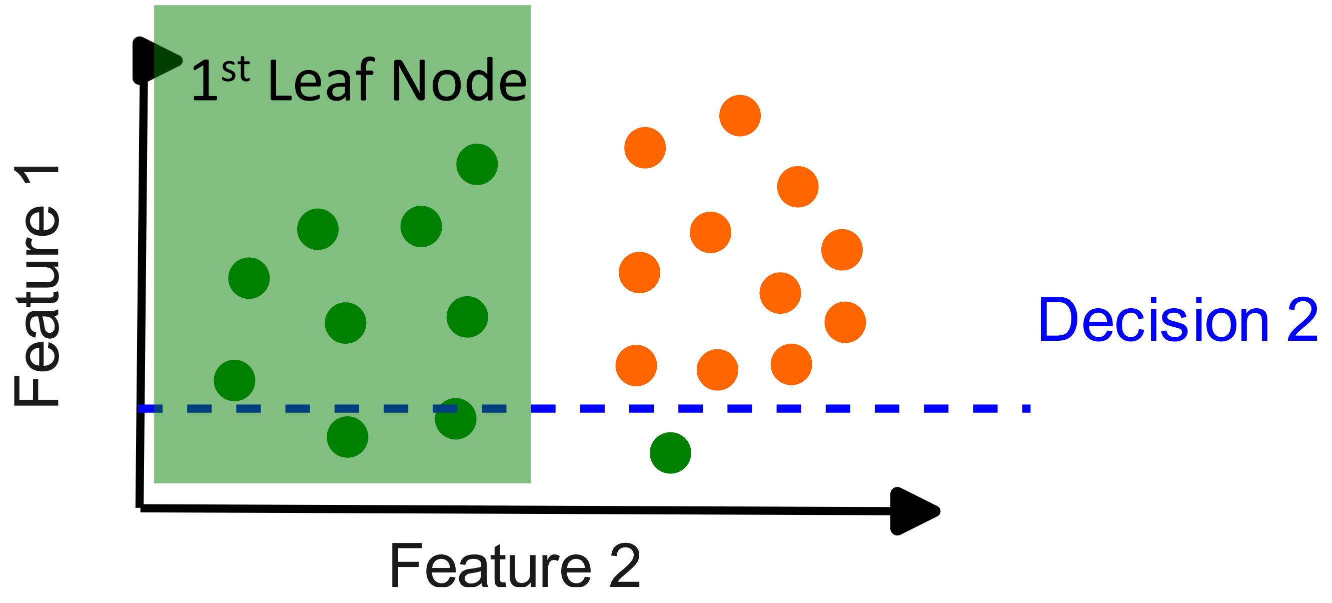
# Decision Trees – Making a Decision

- Now we decided to select on "Feature 1".

- What is the Entropy and Gini of this 2nd decision?

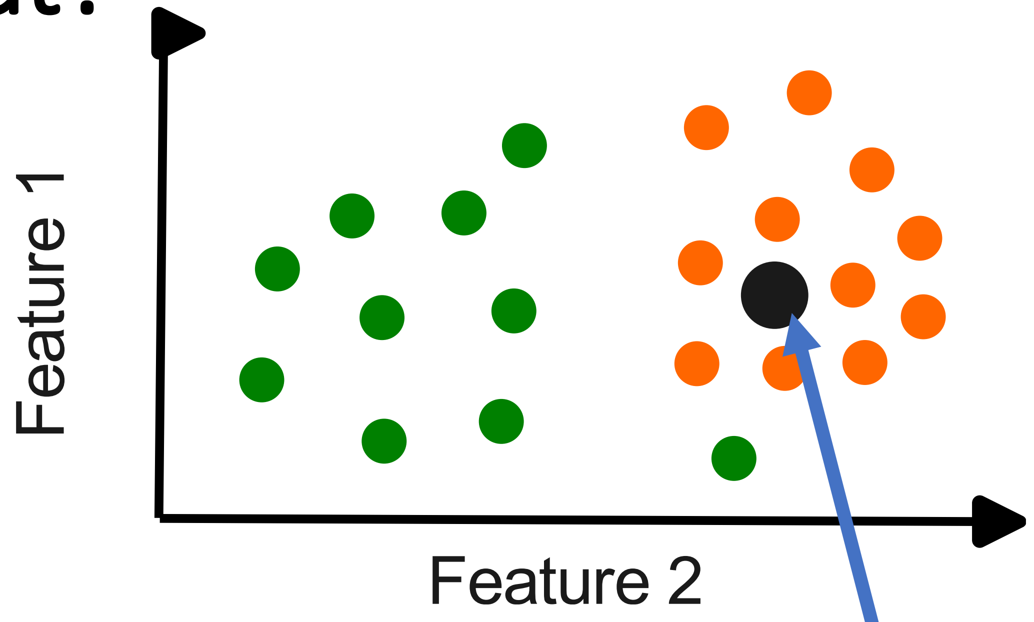
- $E = 0 \ , \ Gini = 0$

- For this dataset we just need 2 decisions based on 2 features to ***classify*** the data into pure samples.



# Decision Trees – Now What?

- We now have a trained classifier on the data.



- This model can be used to make classification decisions on new events

- Input might be new_event(Feature1, Feature2), output is green or orange.

## Decision Trees – Regression (an aside)

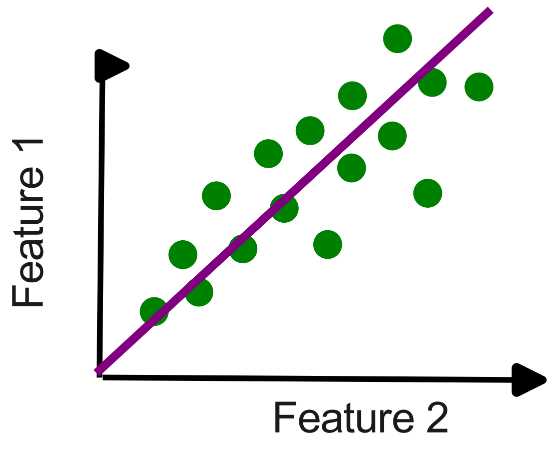
- Just discussed example of using a Decision Tree to make a classification.
- This uses tagged data to train a model to decide on the class for an event based on its features.

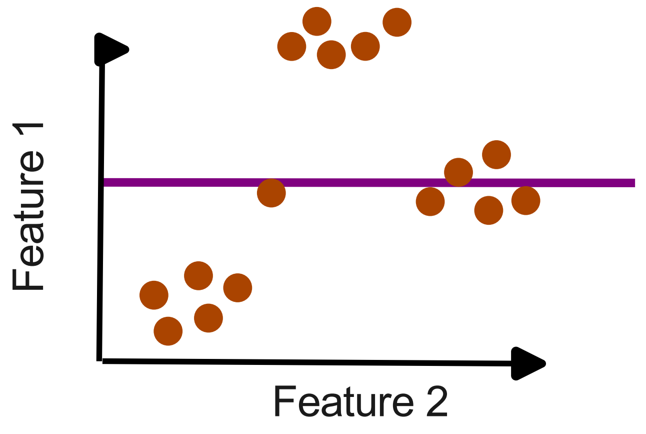
- Decision Trees can also be used to build ***Regression*** models.
- These models are used to generate of a real feature value based on other features.

- These sort of models are used in industry for deciding things like, tax bands, insurance risk, health insurance cost, etc…

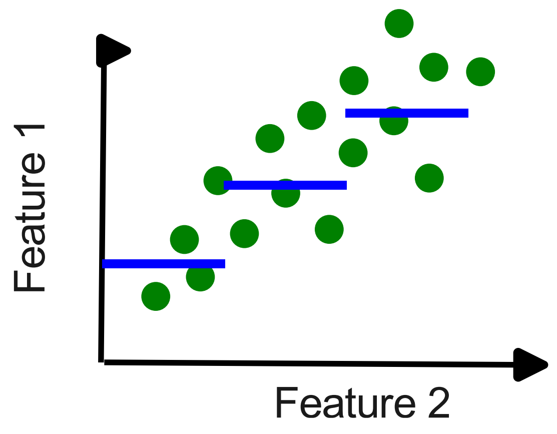
# Decision Trees – Regression (an aside)

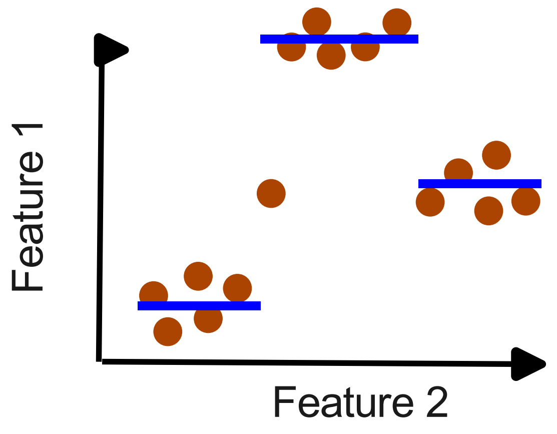
Best Fits from a
Linear Regression:





Best Fits from a
Decision Tree Regression:





## Decision Trees – Real Data

I encourage **everyone** to work through the ‘***Lecture2.ipynb***’ notebook.

This notebook uses the iris dataset based on measurements from multiple species of iris flowers. This is a common freely available dataset used to demonstrate ML fitting to data.

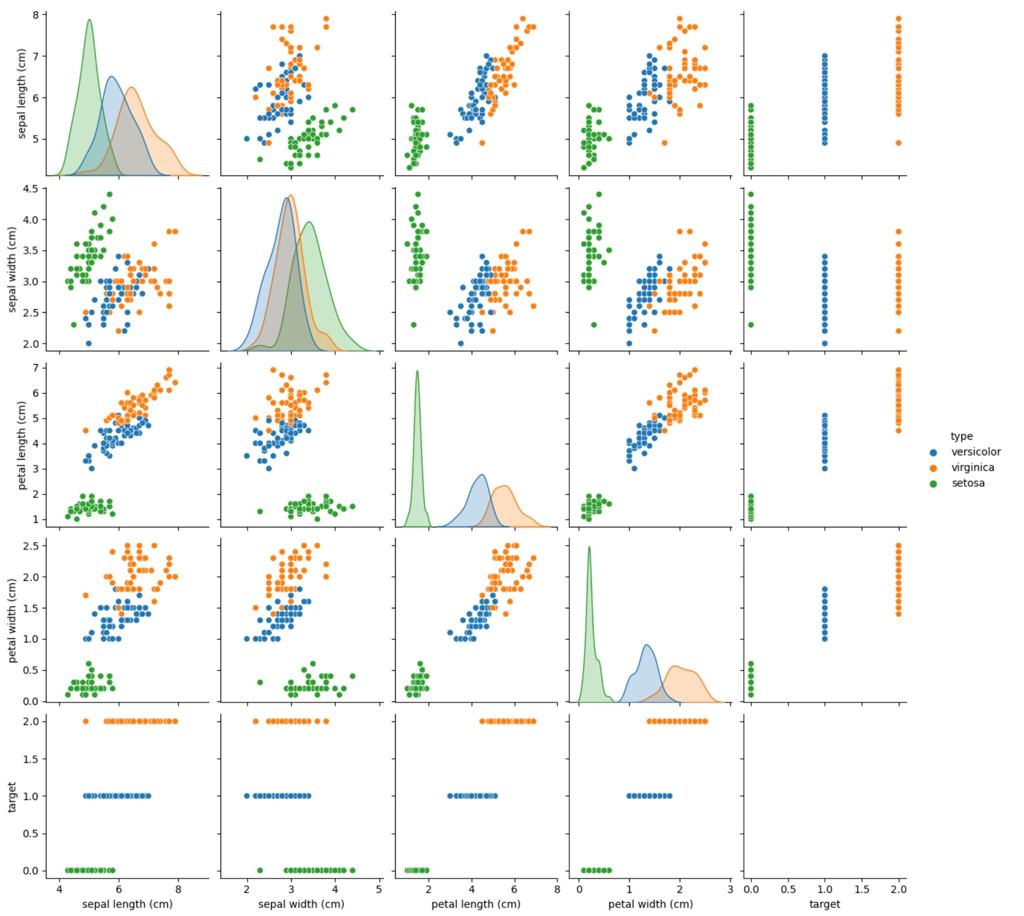
This notebook uses a Decision Tree to make a set decisions to classify flower species based on features such as petal length…

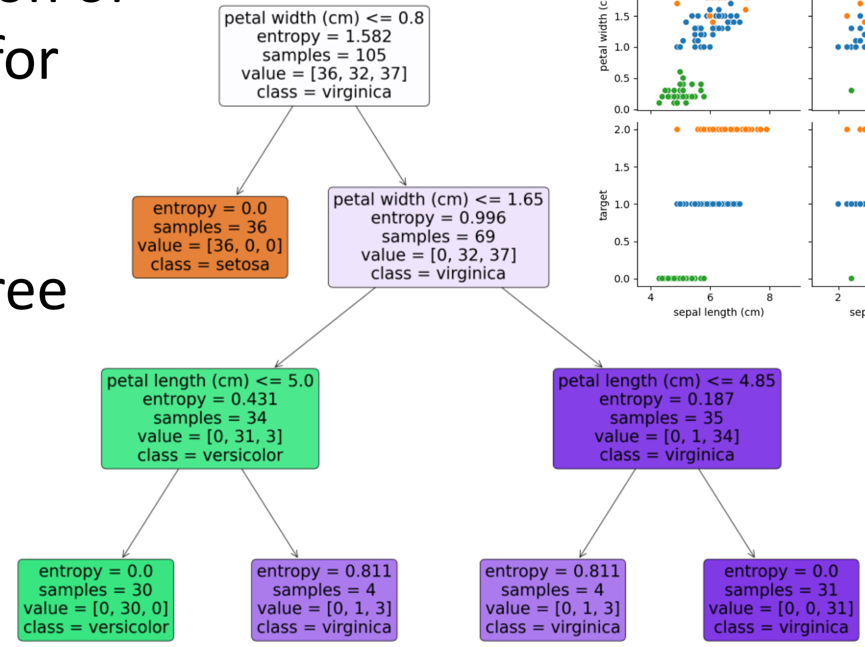
# Decision Trees – Real Data

One of the most important parts before fitting a model to data is to understand the distribution of data in the training dataset.

On the right is the distribution of pairs of features measured for each species of iris flower.

Below is the final decision tree to identify species based on their features.





## Decision/Random – Potential Problems

Decision Tree examples in this course will likely be focussed on slightly small or simplified datasets.

Real world datasets are often many orders of magnitude bigger. Ideally with 10s of 100s of features to be able to fit to the data.

With that in mind a real-world Decision Tree could easily grow in depth to 100s of decisions if allowed to over-train.

## Decision/Random – Potential Problems

Decision Trees by design will try to run until all the “**branches**” are terminated by “**leaves**”.

This sounds good naively but often leads to over-trained models based on the (*finite*!) training data which doesn’t reflect reality well.

Simplest way to reduce this risk is to tell a model to stop after a given number of branches in depth.

This offers *a solution* to the problem but does not guarantee it’s the optimal solution.

# Decision/Random – Potential Problems

- **Suitability**

  As discussed, suitability is best checked by *visualizing the results* of the decision tree vs data.
  Alternatively testing the performance of multiple models shows which performs better.

- **Over-Optimization**

  A common problem with Decision Trees is over-optimization.
  We can understand how over-optimized a trained model is by making predictions and comparing the results to a test labelled data (sub)set. This however doesn't fix the problem.

  Common solutions to fixing this problem come in the form of **Bagging** and **Boosting**.

- **Model Stability**

  Not discussed much so far, Decision Trees tend to be sensitive to the dataset they're being trained on. Small changes in the dataset can cause the model to change more than expected/desirable.

  This can be a symptom of either the distribution of training data, or the model being over-trained.

  Both **Bagging** and **Boosting** can help to improve the model stability.

## Bagging Decision Trees

Bagging is a process for improving a decision tree’s stability and to optimize its performance.

The general process is to generate multiple fake datasets from the real data and then fit each one with a unique Decision Tree and take the average.

The process for generating a fake dataset is called ‘selection with replacement’. The new dataset is generated by randomly selecting data points in the original set.

One of the consequences of this is that a *real single event* can appear *multiple times* in a fake dataset.

The principle behind this is that averaging over many fits is more stable and over-fits less on outlying random events.

# Bagging Decision Trees – *Random Forest*

***Random Forests*** are technically a special case of the **Bagging** approach to improve the quality of a Decision Tree.

With ***Random Forests*** the same full training dataset is always used, but multiple Decision Trees are constructed which ignore certain features within the dataset.

This is <u>one of the most widely used</u> bagging methods.

It has several advantages such as better stability when fitting the dataset. However, this is one of the more difficult bagging approaches to implement due to its complexity.

## Boosted Decision Trees

**Boosting** is another form of Decision Tree optimization.

Typically, these take the approach of:

1. Perform a fit with a model to the data.
2. Use this model to make predictions of the input dataset.
3. Calculate error in predictions.
4. Construct residual weights based upon errors.
5. Make a new model that considers the residual weights.
6. Return to Step 2. until some limit is reached.
7. Take final model.

Intermediate trees are referred to as weak learners and the final tree is referred to as a strong learner.

# Boosted Decision Trees

**Boosting** algorithms build decision trees 1 at a time in an iterative way rather than **Bagging** which can construct multiple trees and perform fits in parallel.

3 of the most popular Boosting algorithms are:

**AdaBoost**, **GradientBoost** and **XgBoost**

**AdaBoost** is focussed primarily on using residuals to correct for mis-identification in the results of the decision tree.

**GradientBoost** and **XgBoost** perform progressive fits trying to reduce the size of the residual errors with each consecutive tree.

**XgBoost** is the most popular algorithm in the field of ML and introduces a concept known as *regularization* and many other features. This makes it the best, but is also the most difficult to explain/conceptualise…

## Workshop – Tomorrow

The workshop Notebook is a chance to do a basic analysis with a DecisionTree model using SciKit Learn using a Jupyter notebook.

The dataset in question is a collection of data reviews and other features different types of wine.

An important part of this is to understand plotting and displaying features from this dataset. Then, perform a DecisionTree fit to the data and decide if the DecisionTree is a good fit to the data or not.