Machine Learning with Python

Thiago S Mosqueiro

http://thmosqueiro.vandroiy.com/MachineLearning_CollW

UCLA QCBio Collaboratory

Day 1 – 11/28/2017

http://qcb.ucla.edu/collaboratory/

What is Machine Learning?

"Field of study that gives computers the ability to learn without being explicitly programmed."

Arthur Samuel, 1959

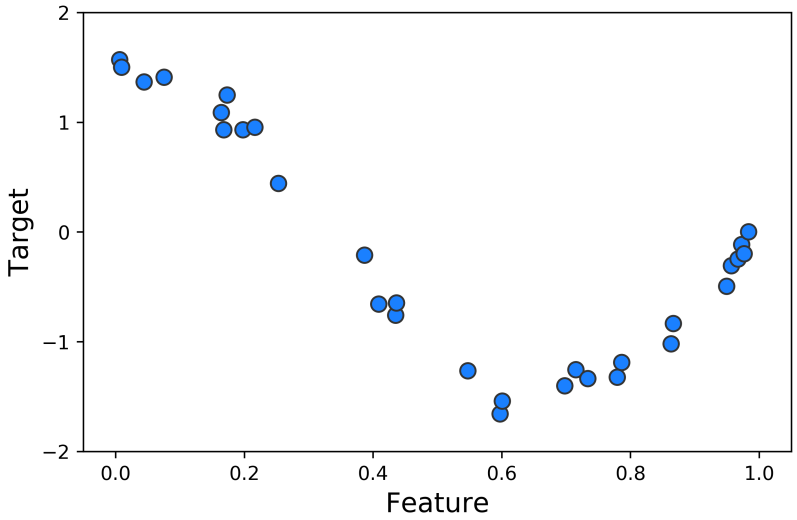
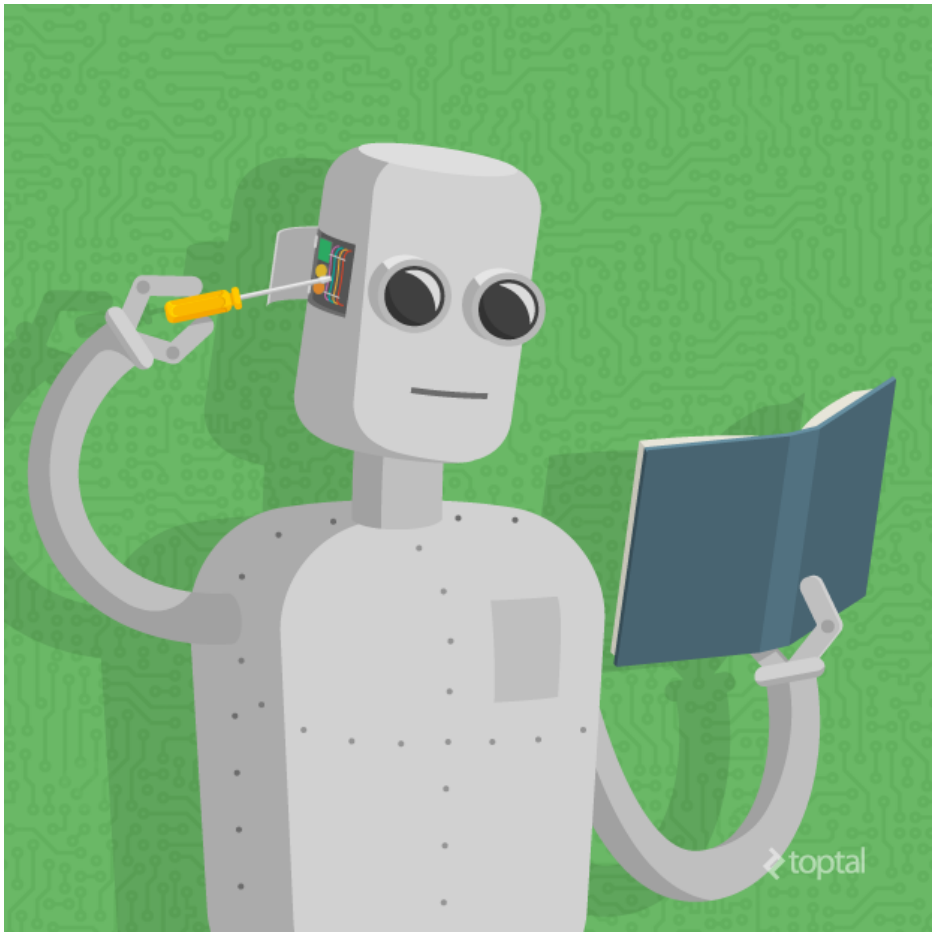


Image from https://www.toptal.com/



What is Machine Learning?

"Field of study that gives computers the ability to learn without being explicitly programmed."

Arthur Samuel, 1959

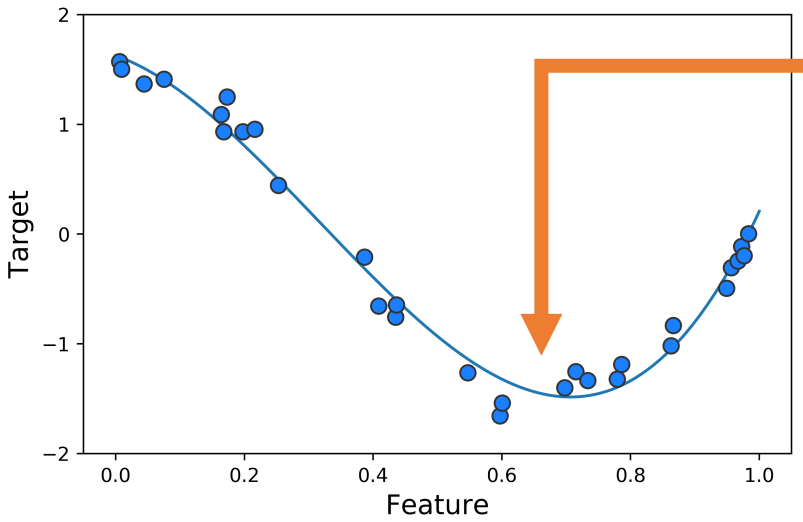
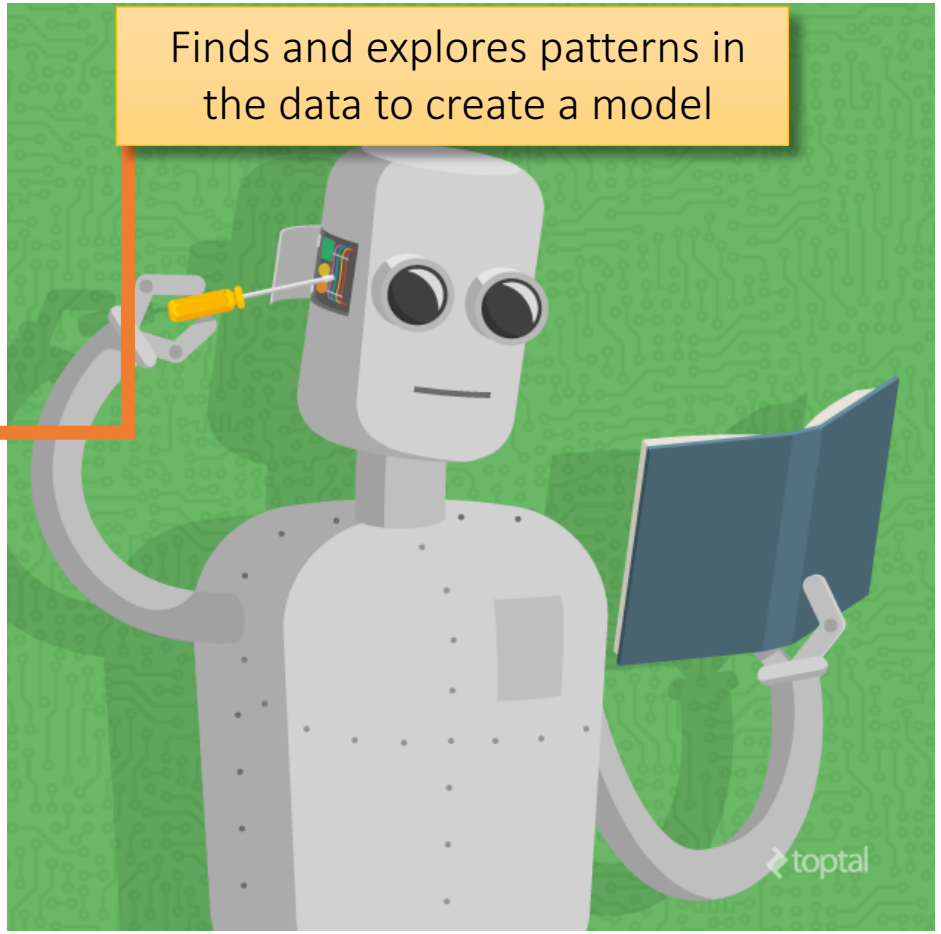
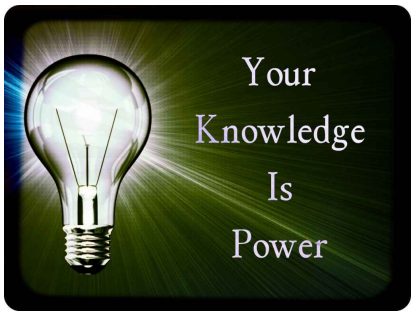


Image from https://www.toptal.com/



What is our goal here?

Our goal is to provide you with the basic information to allow you to become independent and use online resources to your favor!

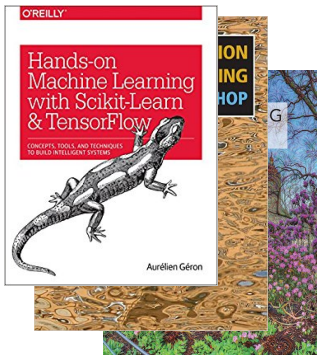


+

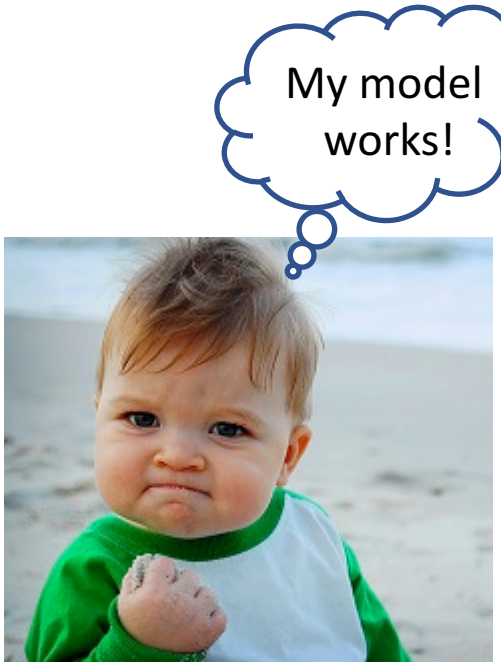
Google

stackoverflow

CrossValidated



=



Workshop Overview

Day 1 – Fundamentals and Motivation

- Nano-review of Python & Introducing Jupyter
- Importing and handling datasets
- Introducing Scikit-Learn

Day 2 – Classification & Cross validation

- Learning to choose and employ models from Scikit-Learn
- Assessing the performance of your models
- Properly cross-validating

Day 3 – Regression and Non-supervised learning

- Linear and non-linear regression
- Cluster analysis and hierarchical clustering
- Final remarks

Day1- Contents and overview

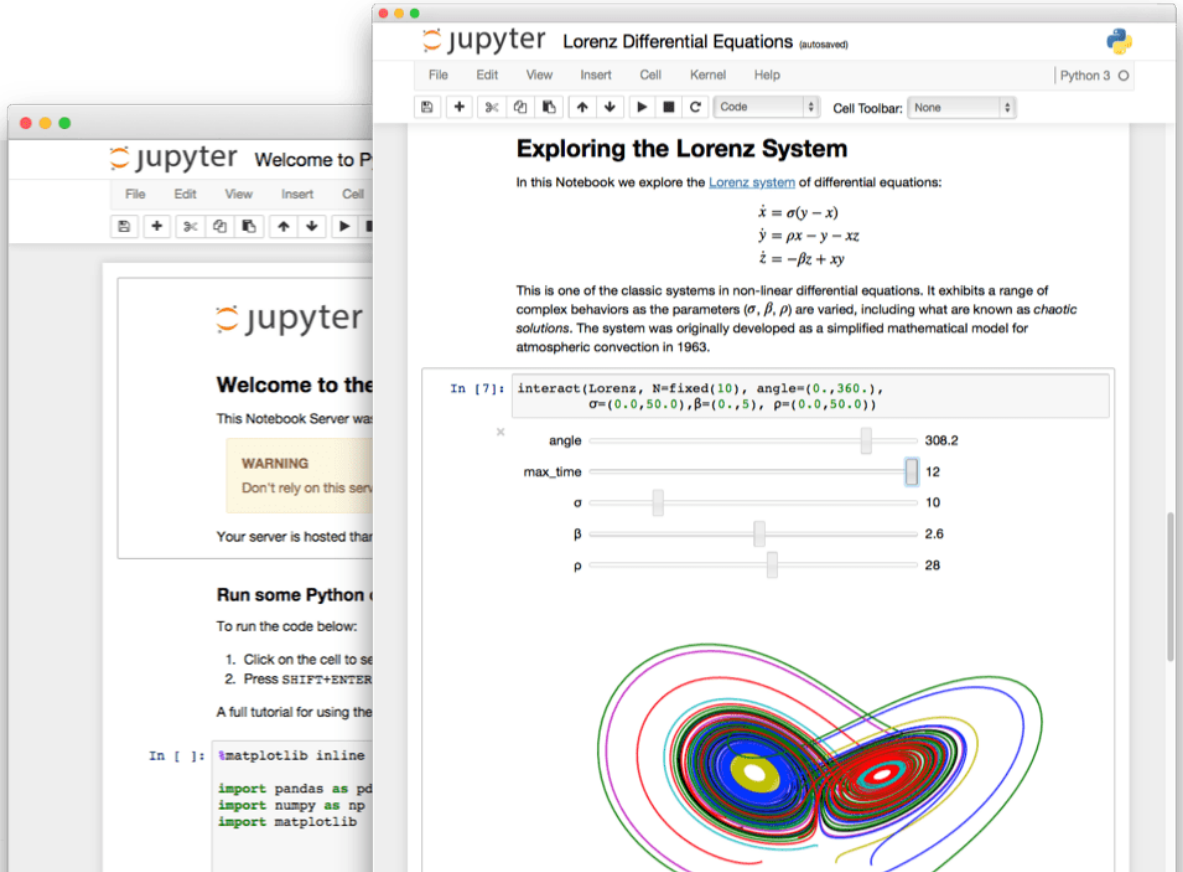
- Nano-review of python
- Exploring NumPy and Matplotlib
- Overview of Machine Learning
- Getting started with Scikit-Learn
- In-class Practice

Nano-review of Python and Jupyter Notebooks

Jupyter notebook

Jupyter Notebook is an open-source application that allows you to create and share documents that contain live code and hyper-text

Anaconda ships with Jupyter Notebooks by default

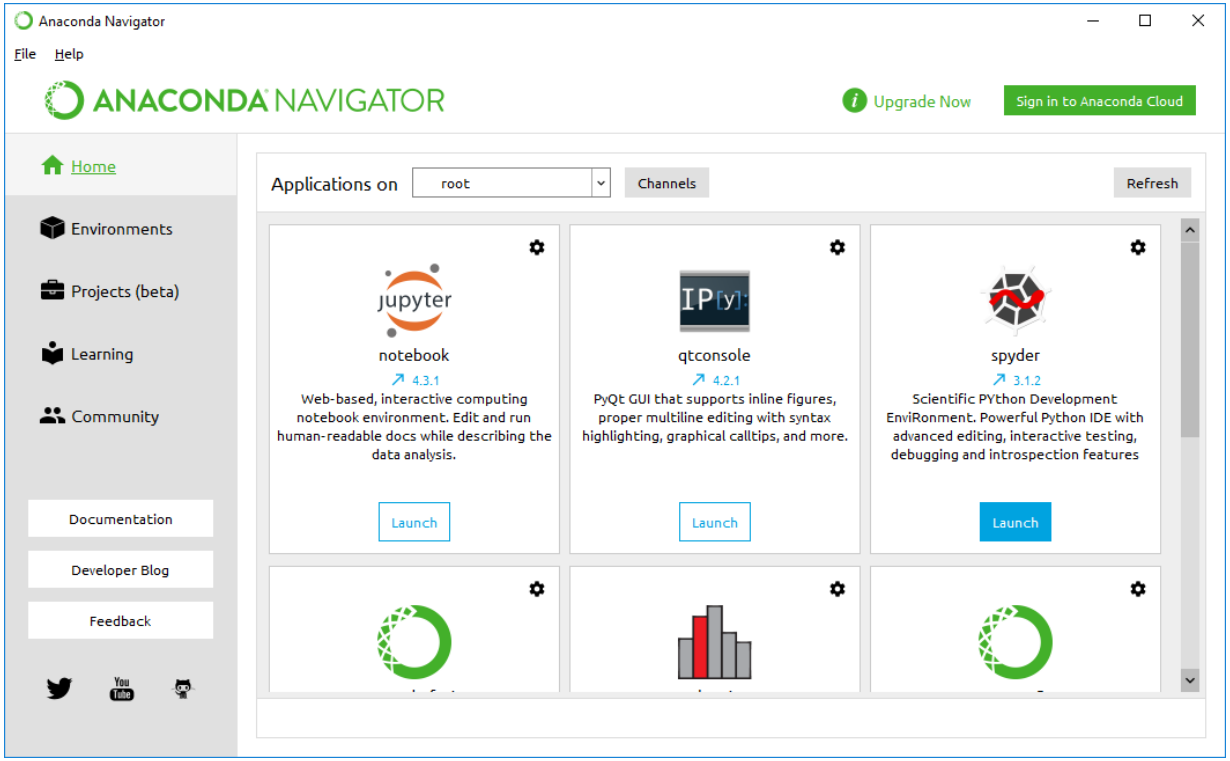


Let's get started

Open the Anaconda Navigator by…

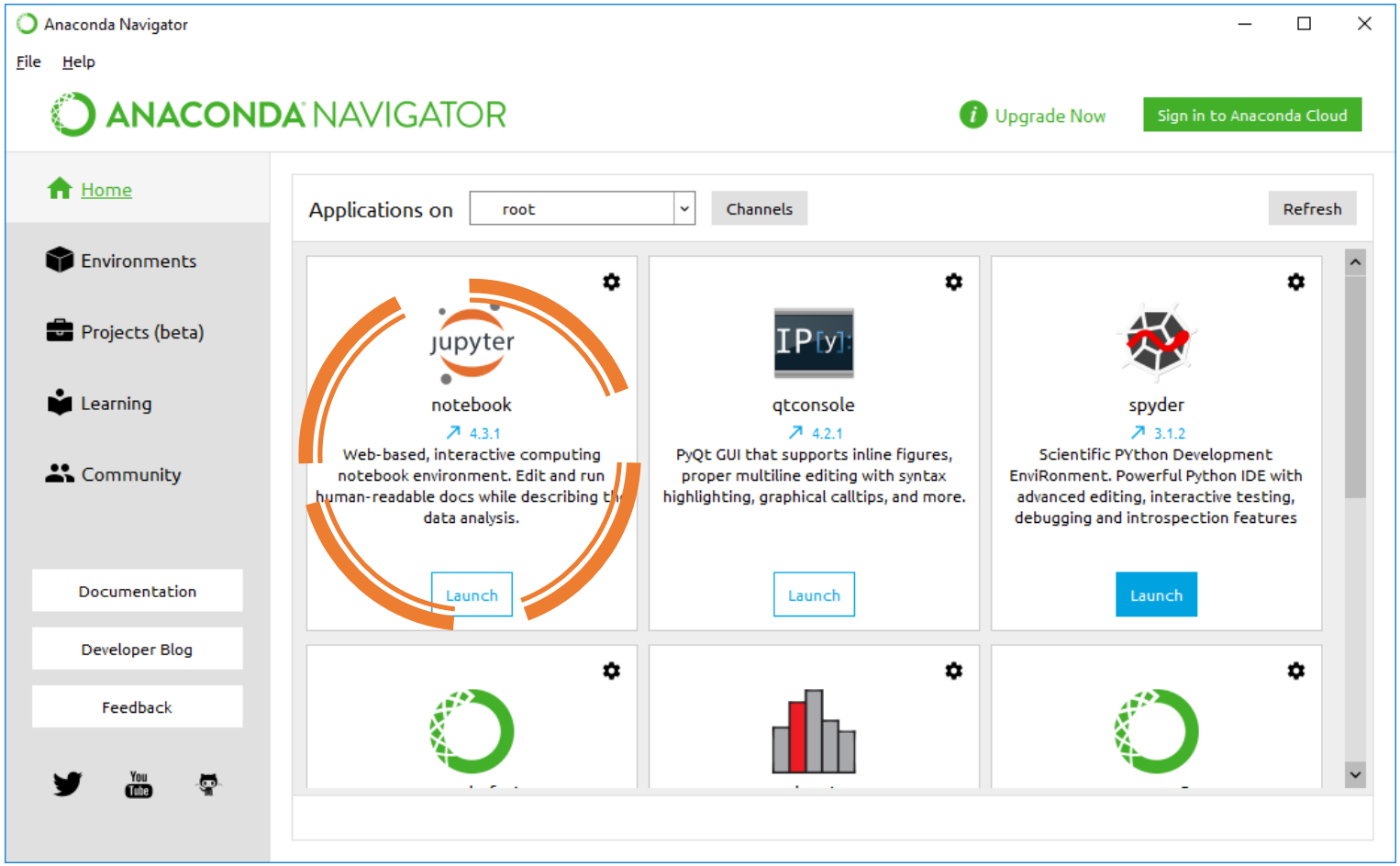
- MAC: Hit command+<space bar> and type "Anaconda"
- WINDOWS: Look for Anaconda on your start menu

It should look like this:



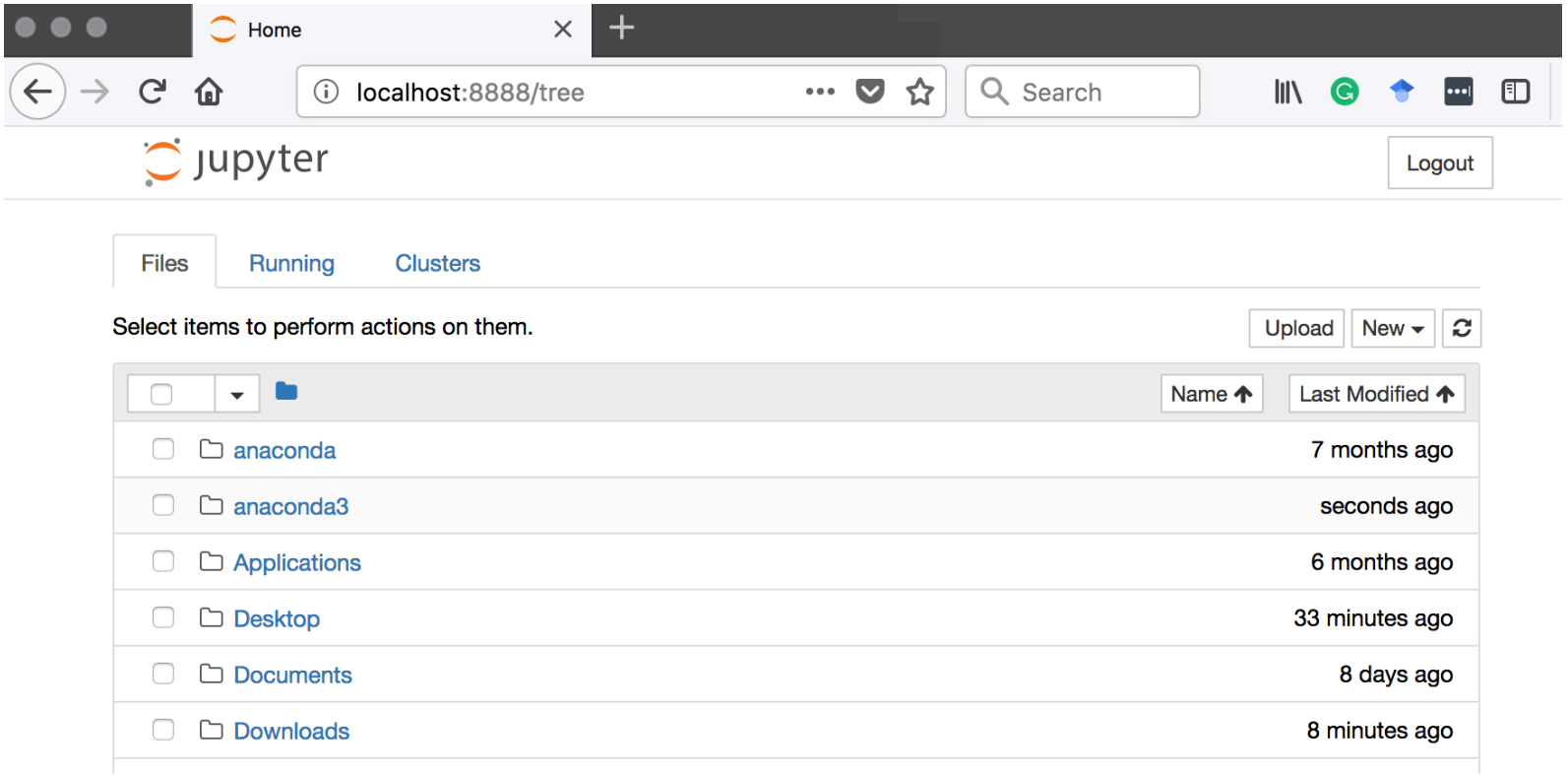
Let's get started

Then click on the Jupyter Notebook item

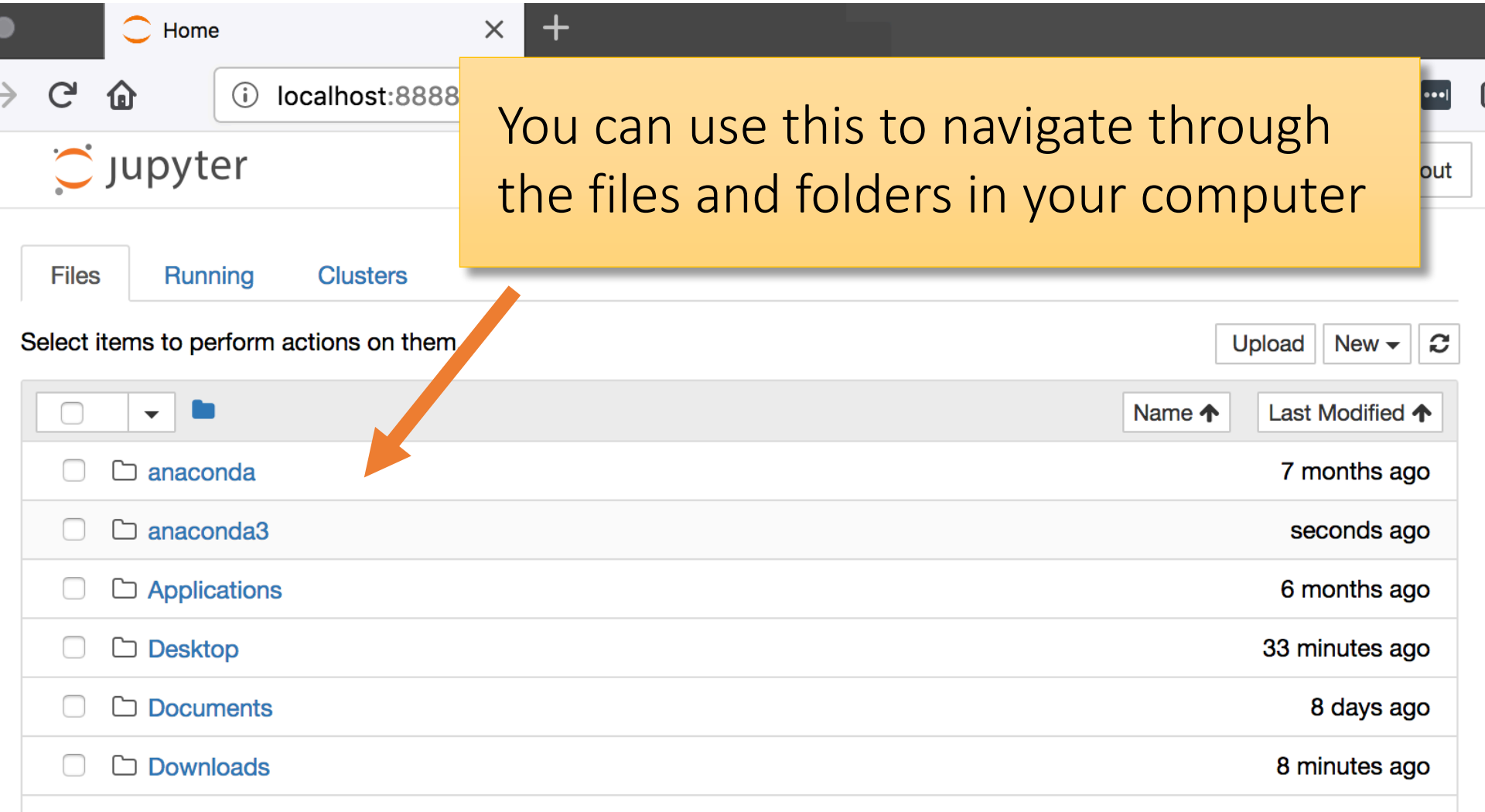


Jupyter Dashboard

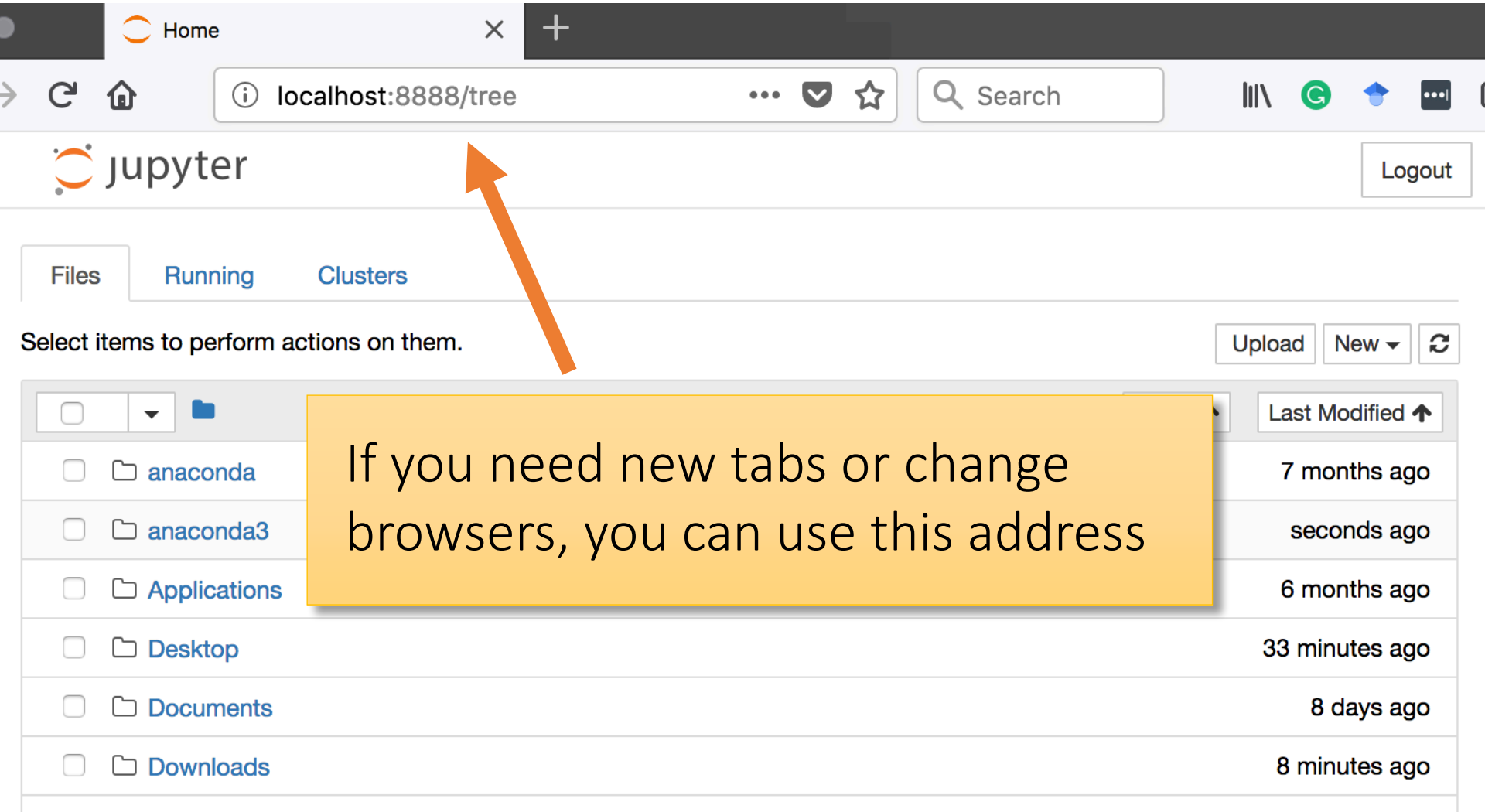
After a few moments, you should see your browser open something similar to this:



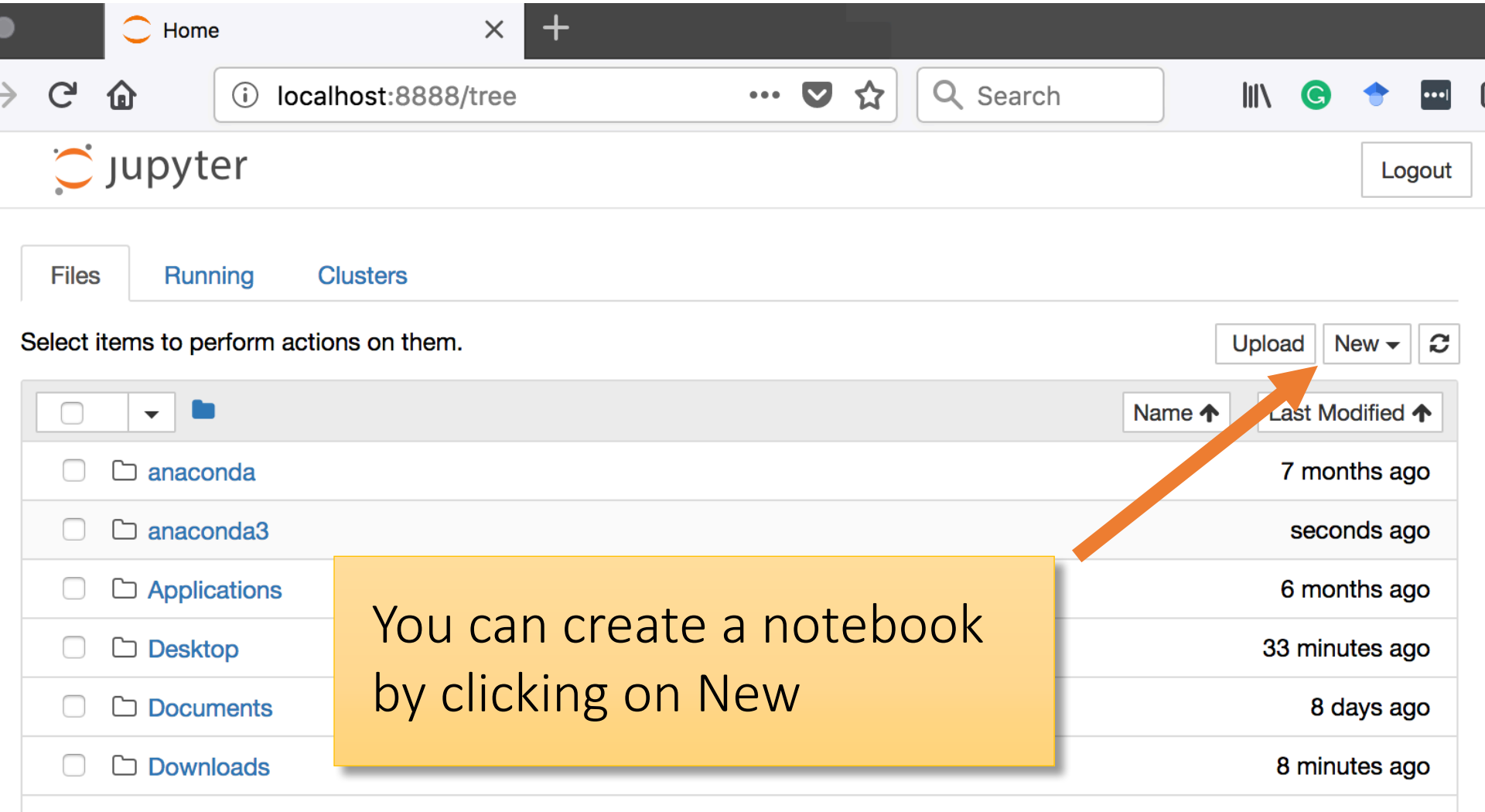
How to use it…


Home
localhost:8888
jupyter
You can use this to navigate through the files and folders in your computer
Files
Running
Clusters
Select items to perform actions on them
Upload
New
Name
Last Modified
anaconda
7 months ago
anaconda3
seconds ago
Applications
6 months ago
Desktop
33 minutes ago
Documents
8 days ago
Downloads
8 minutes ago

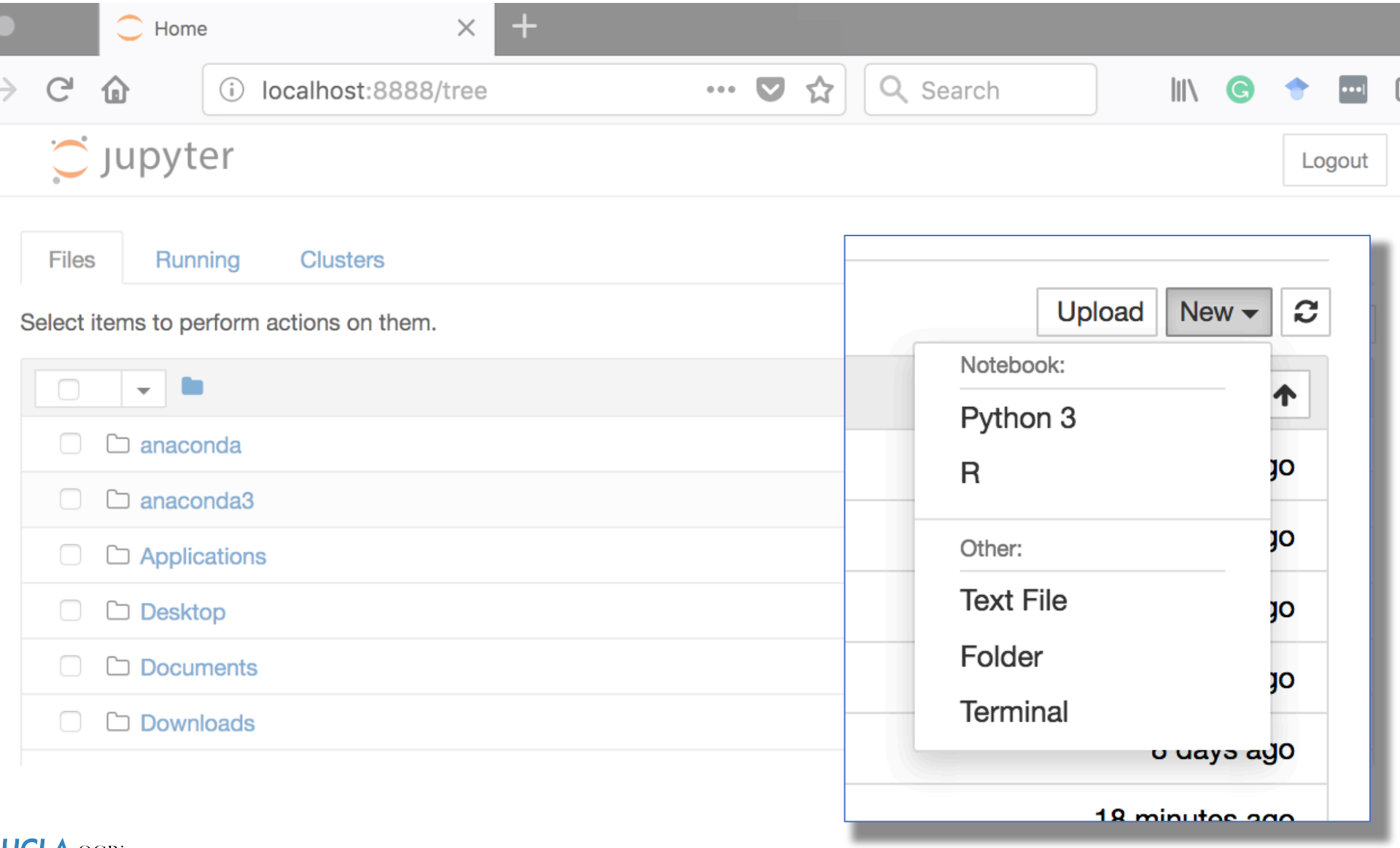
How to use it…


Home
localhost:8888/tree
Search
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Last Modified
anaconda
7 months ago
anaconda3
seconds ago
Applications
6 months ago
Desktop
33 minutes ago
Documents
8 days ago
Downloads
8 minutes ago
If you need new tabs or change browsers, you can use this address

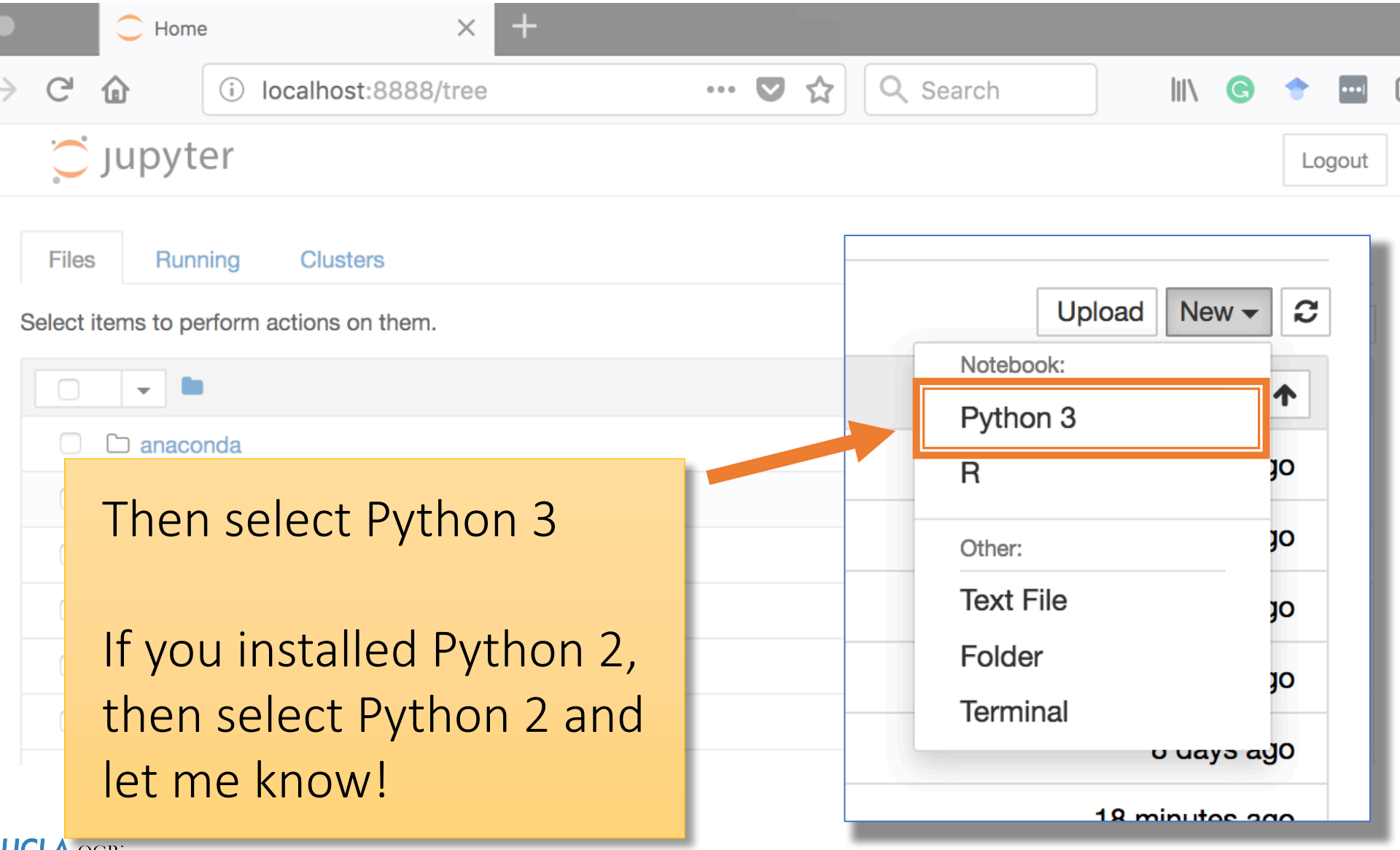
How to use it…


Home
localhost:8888/tree
Search
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Name
Last Modified
anaconda
7 months ago
anaconda3
seconds ago
Applications
6 months ago
Desktop
33 minutes ago
Documents
8 days ago
Downloads
8 minutes ago
You can create a notebook by clicking on New

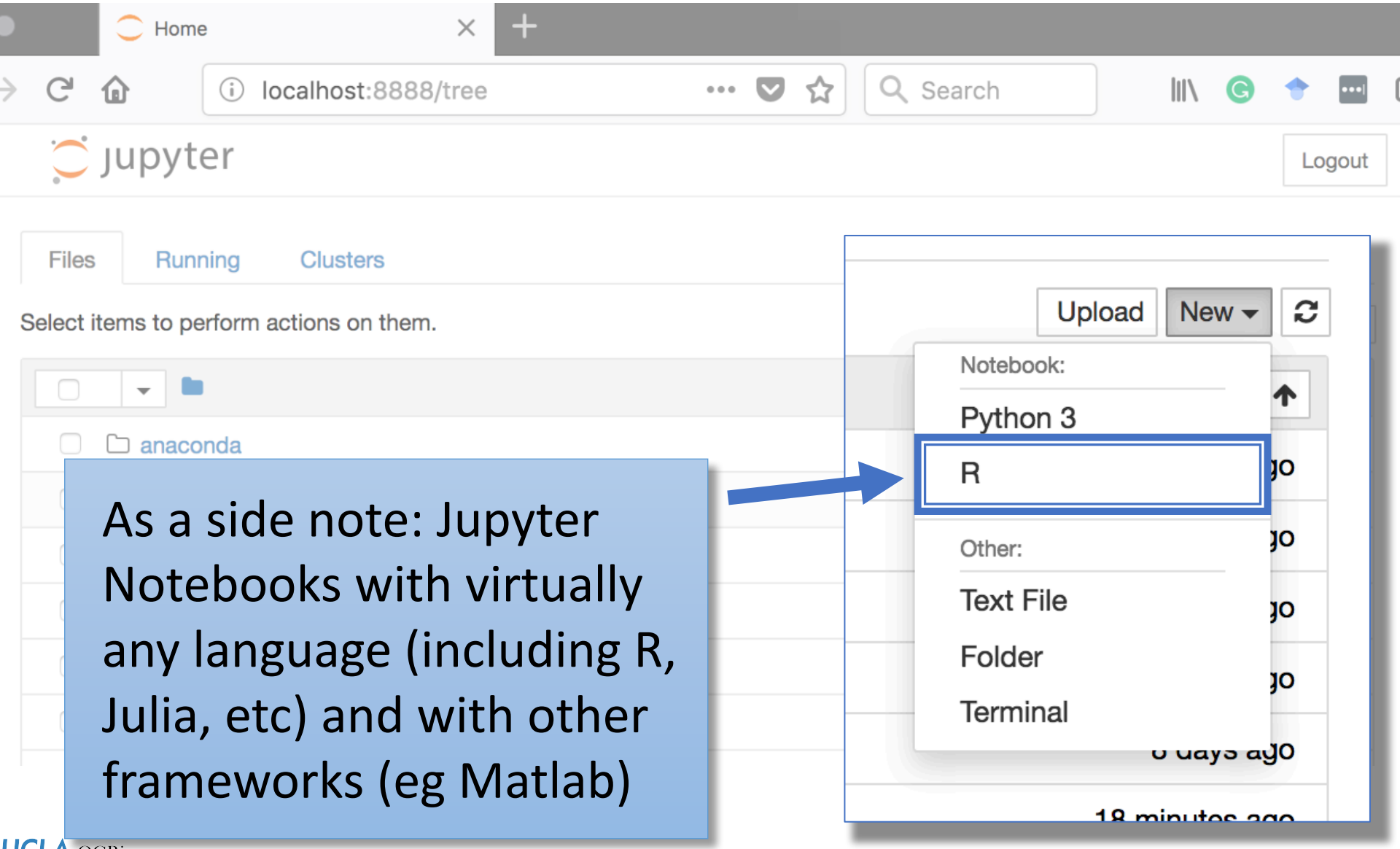
How to use it…


Home
localhost:8888/tree
Search
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
anaconda
anaconda3
Applications
Desktop
Documents
Downloads
Upload
New
Notebook:
Python 3
R
Other:
Text File
Folder
Terminal

How to use it…


Home
localhost:8888/tree
Search
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
anaconda
Upload
New
Notebook:
Python 3
R
Other:
Text File
Folder
Terminal
Then select Python 3
If you installed Python 2, then select Python 2 and let me know!

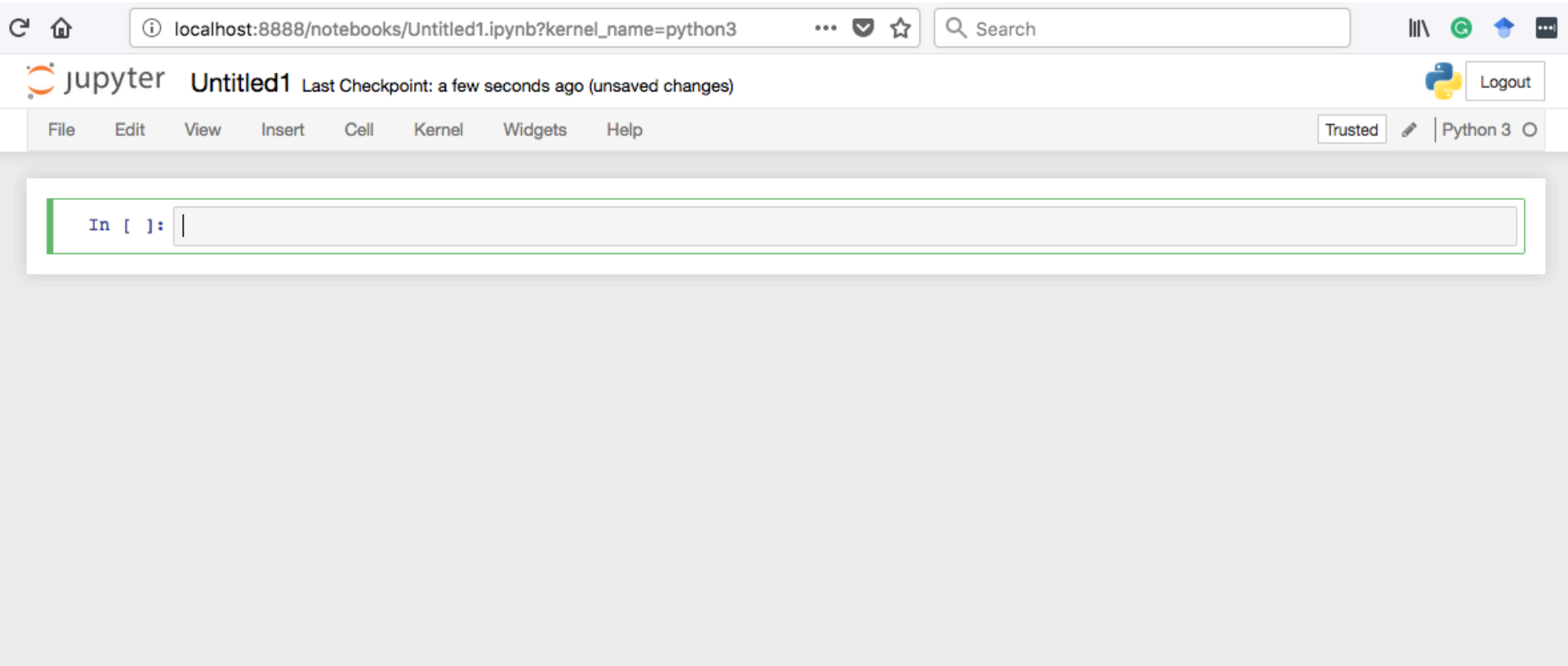
How to use it…



As a side note: Jupyter Notebooks with virtually any language (including R, Julia, etc) and with other frameworks (eg Matlab)

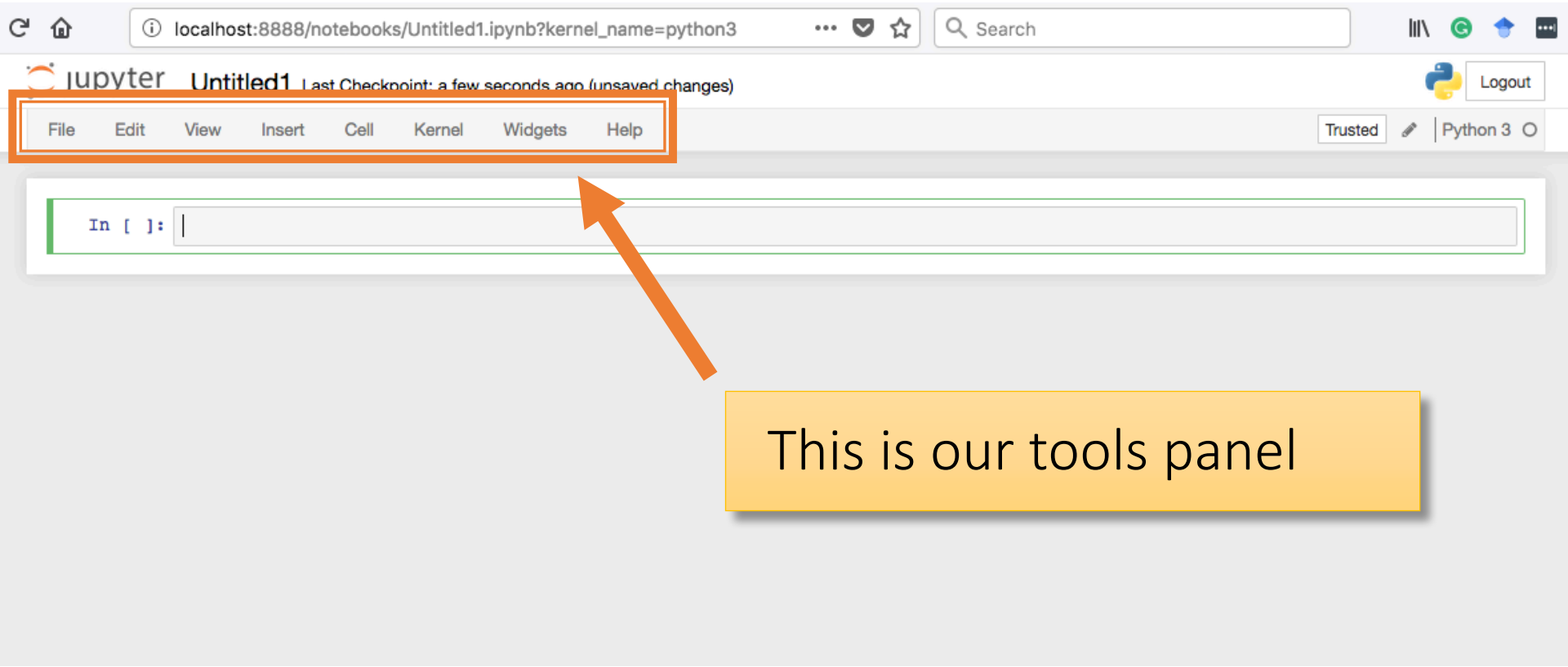
Your first notebook

After you select the language, you should see a "blank" notebook on your browser:



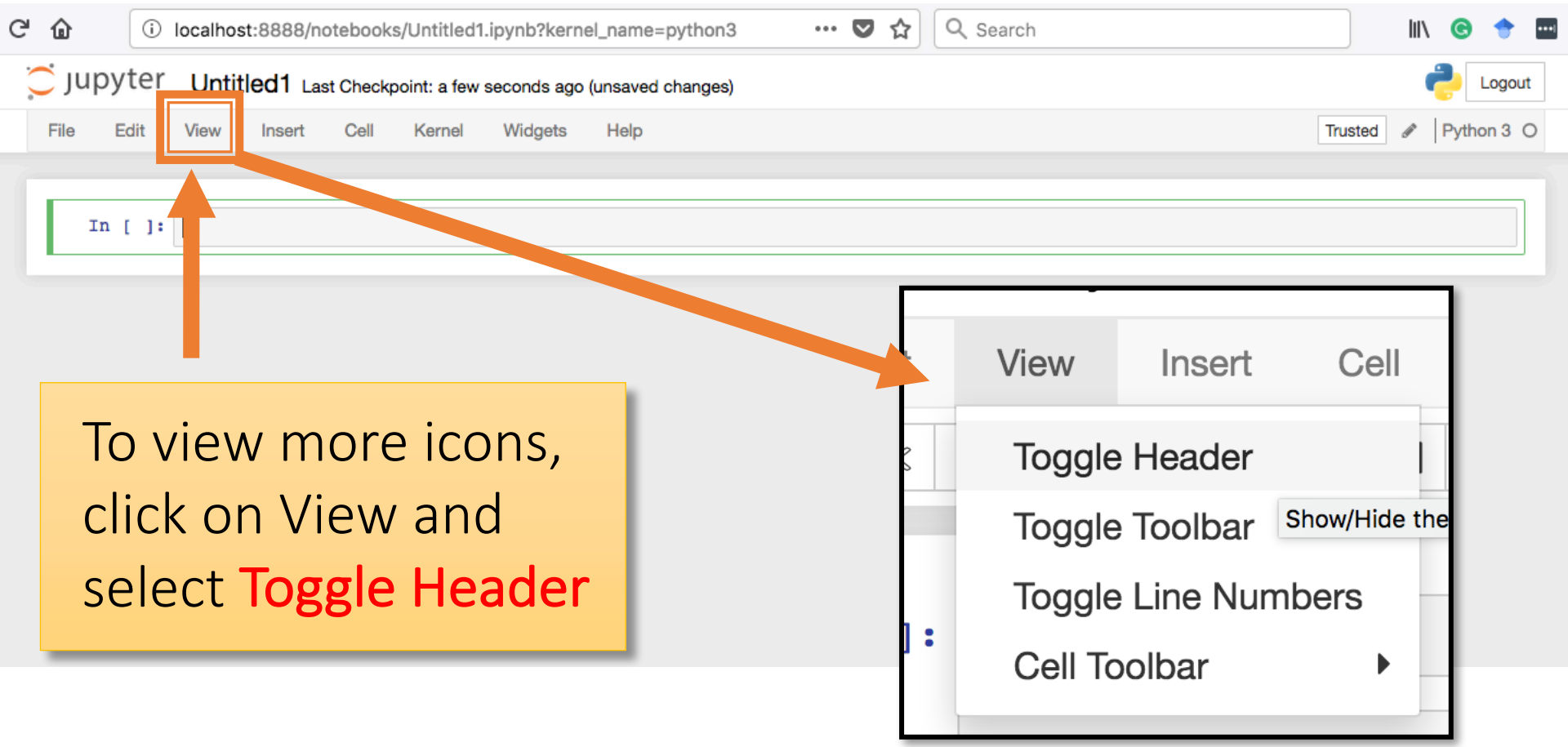
Your first notebook

After you select the language, you should see a "blank" notebook on your browser:



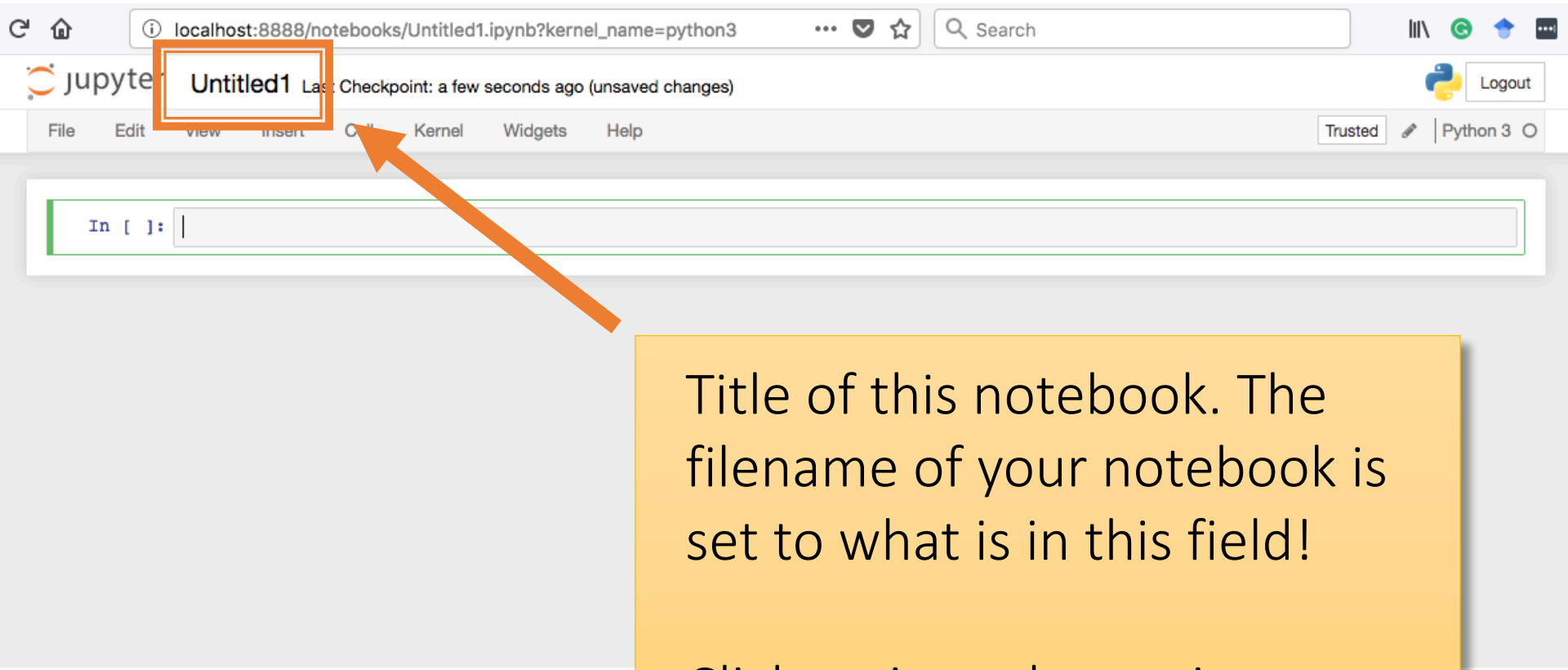
Your first notebook

After you select the language, you should see a "blank" notebook on your browser:



Your first notebook

After you select the language, you should see a "blank" notebook on your browser:

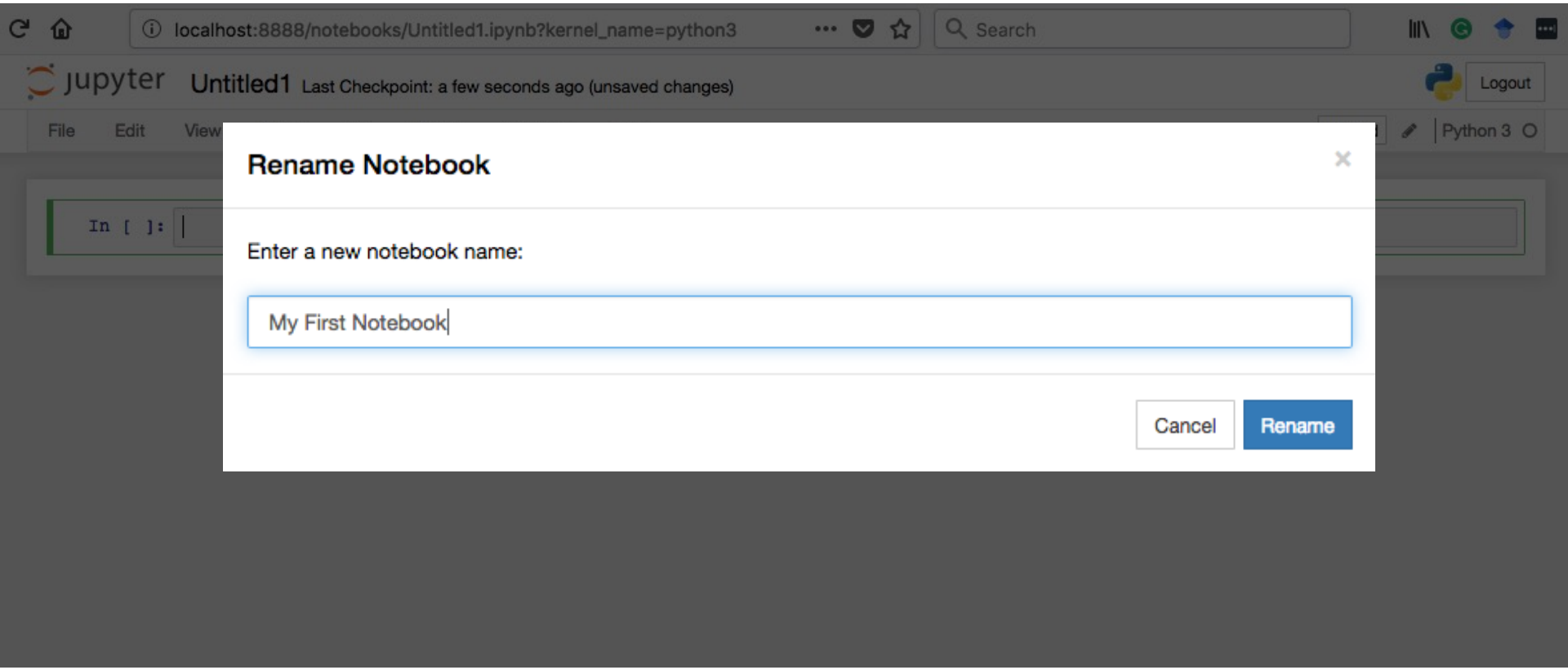


Title of this notebook. The filename of your notebook is set to what is in this field!

Click on it to change its name.

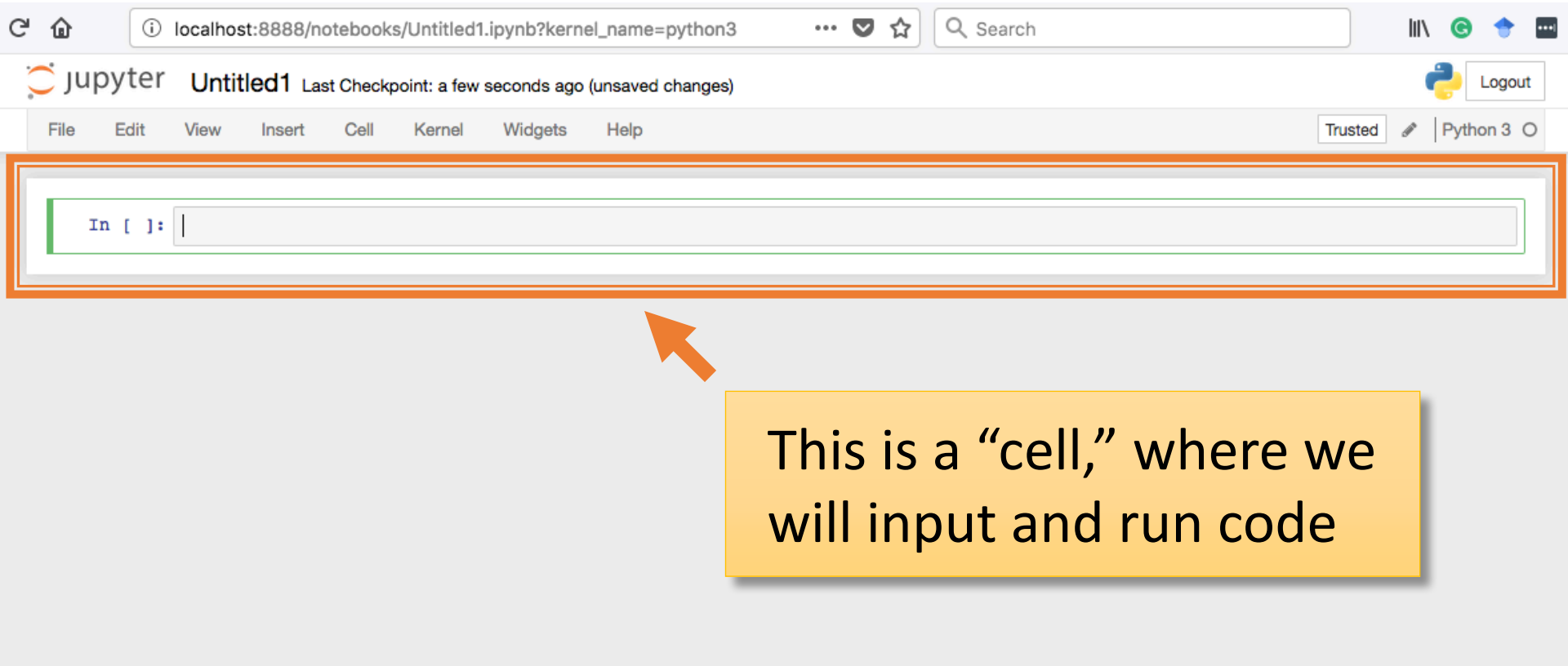
Your first notebook

After you select the language, you should see a "blank" notebook on your browser:



Your first notebook

After you select the language, you should see a "blank" notebook on your browser:



This is a "cell," where we will input and run code

Inputing code into a cell

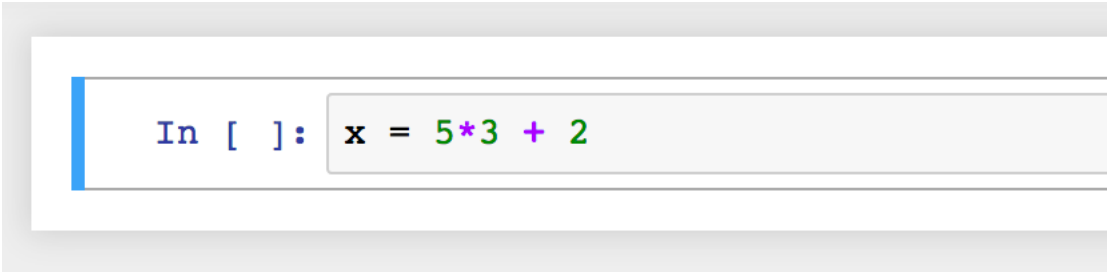
To run some Python code, all you need is to write it to a cell and run it.

Let's try something simple. Click on the only cell available on your screen and write down the following:

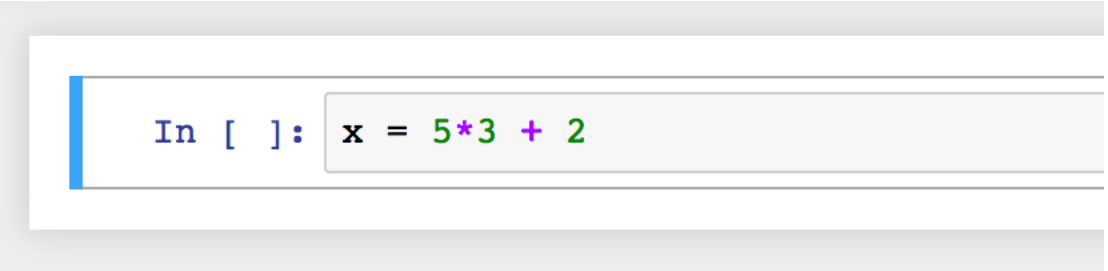
```
x = 5*3 + 2
```

This should save the number 17 in the variable x.

Try writing this simple Python code into the cell.



Syntax highlighting



Note that Jupyter Notebook will automatically use different colors for different elements.

This is intentional, and helps programmers to see patterns in the code. You may not see the value of it now, but with experience **syntax highlighting** will be your best friend…

For instance:

- Numbers are green
- Mathematical operations are purple

Executing the contents of a cell

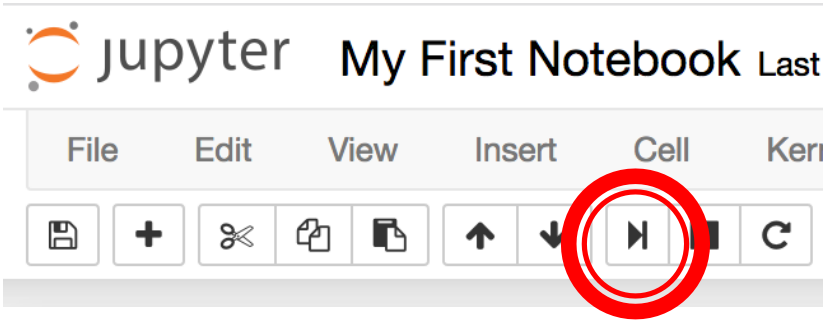
There are two ways to execute the code:

1- Using a shortcut in your keyboard:

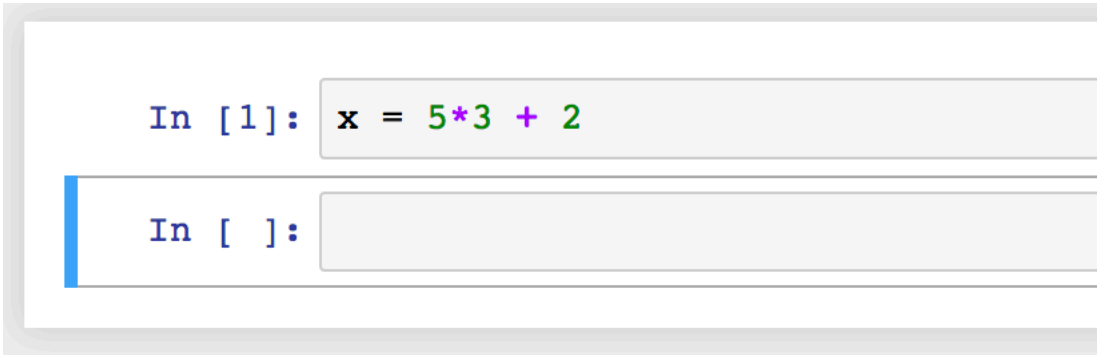
MAC – hit **COMMAND + ENTER**

WINDWS – hit **CONTROL + ENTER**

2- Click on the "run cell" icon



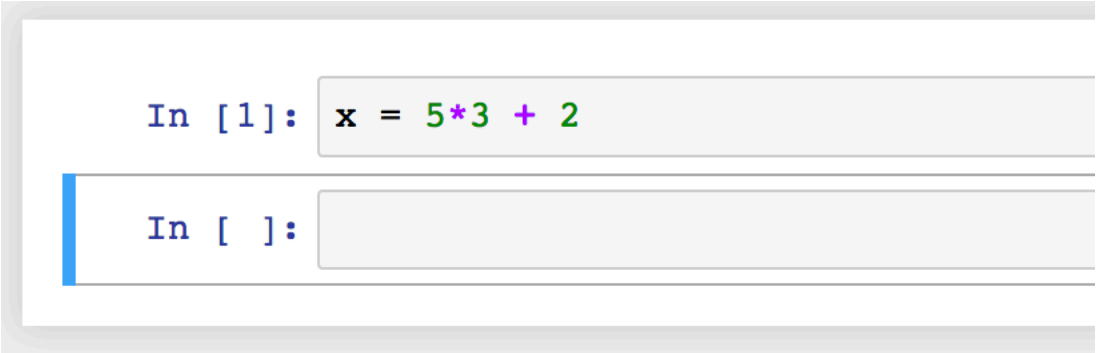
Regardless of how you run it, this is what you should see next:



Executing the contents of a cell

Did it really run anything?

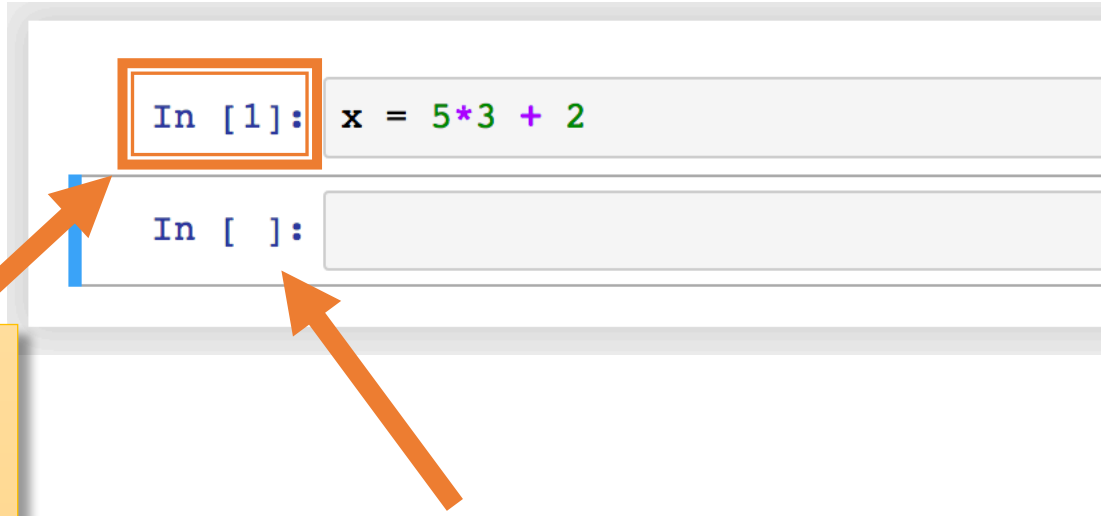
Yes, it did.



Executing the contents of a cell

Did it really run anything?

Yes, it did.



The presence of brackets ([]) with a number indicates that this cell was executed!

This is how a cell that was never executed before look like…

Executing the contents of a cell

You can use more than just one line per cell

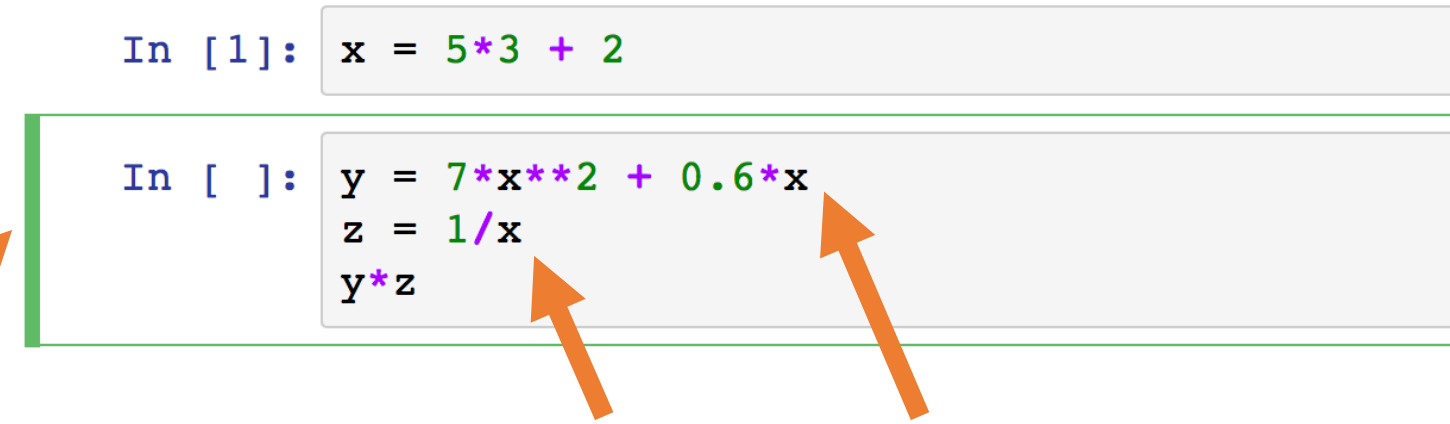
click on the new cell and let's
input the following piece of code:

```
In [1]: x = 5*3 + 2

In [ ]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z
```

Executing the contents of a cell

You can have more than just one line in each cell:

click on the new cell and let's
input the following piece of code:



This is how the selected cell is distinguished

Use ENTER to input as many lines of code necessary, but avoid **interrupting** a line of code in the middle

Output of code

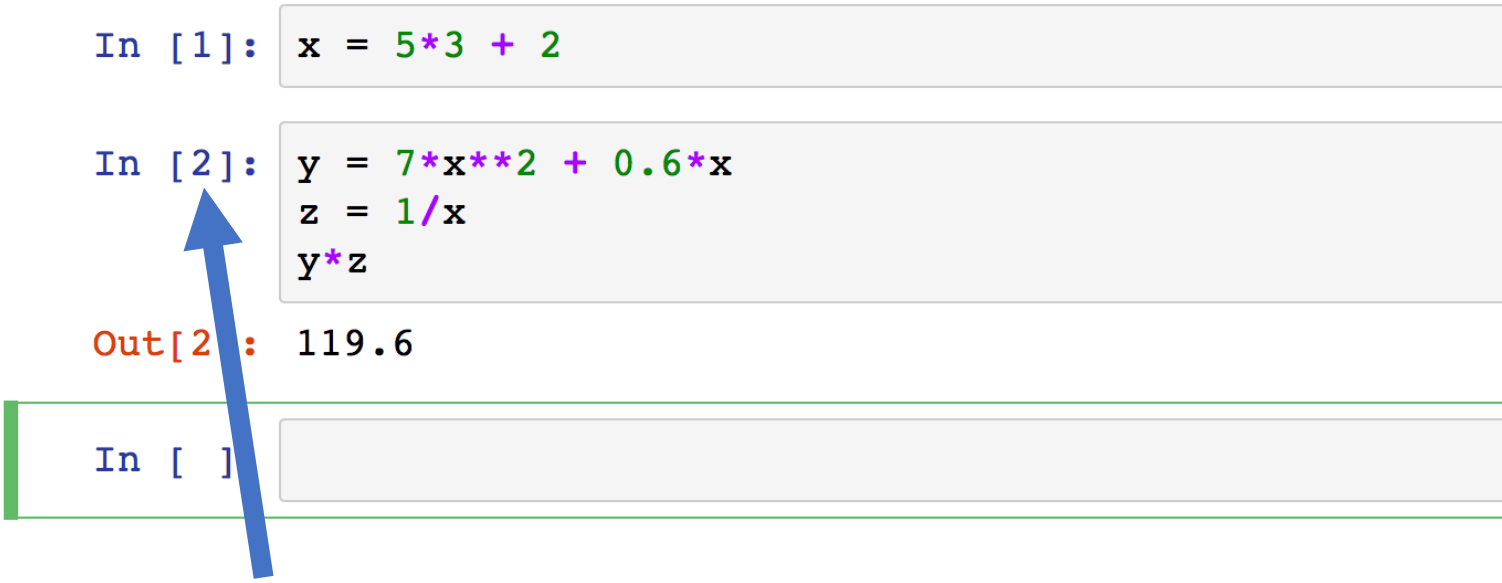
Try and execute this last cell.

Was it any different from our first cell?

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [ ]:
```

Output of code

Try and execute this last cell.
Was it any different from our first cell?



After executed, this new cell shows the number [2]!
These numbers indicate the
order in which cells are executed!

Output of the last line

Try and execute this last cell.
Was it any different from our first cell?

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [ ]:
```

This time, we are also seeing **the result** of the operation `y*z` – We didn't we see anything before?

Output of the last line

Whenever the last line of a cell has no attributions (i.e., = signs!), then Jupyter Notebook will display the result of that last operation.

Let's try it on our third cell!

```
In [3]: 5*3 + 2

Out[3]: 17
```

- Can you think of another way to display something in Python??

Output of the last line

Whenever the last line of a cell has no attributions (i.e., = signs!), then Jupyter Notebook will display the result of that last operation.

Let's try it on our third cell!

```
In [3]: 5*3 + 2

Out[3]: 17
```

- Can you think of another way to display something in Python??

```
In [4]: print(y*z)

        119.6
```

Practice – Python

Using this same notebook, let's try the following:

1. Check if the variable x is indeed 17.
2. Redefine x as a list of all integers between 0 to 10
3. Write a for-loop that prints the square of each of the elements of x
4. Write another for-loop to find the sum of the square of the odd elements in x

Let's solve it together…

1. Check if the variable x is indeed 17.
2. Redefine x as a list of all integers between 0 to 10

```
In [5]: print(x)
        17

In [6]: x = [0,1,2,3,4,5,6,7,8,9,10]
        print(x)
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Keep in mind that lists are not "vectors" as we usually we define in math. For instance:

```
[1,2,3,4] + [3,4,2,1] = [1,2,3,4,3,4,2,1]
```

Let's solve it together…

3. Write a for-loop that prints the square of each of the elements of x

```
In [7]: for element in x:
            print(element,element**2)
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

Let's solve it together…

3. Write a for-loop that prints the square of each of the elements of x

```
In [7]: for element in x:
            print(element,element**2)
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

It is OK if your solution was not EXACTLY like this one.

If you are having problems with for-loops and other Python basic topics, don't give up – we will be reviewing them as we progress.

If anything is not 100% clear, do ask questions!

Let's solve it together…

4. Write another for-loop to find the sum of the square of the odd elements in x

In [8]:
```
sumOfX = 0
for element in x:
    if element % 2 == 1 :
        print(element)
        sumOfX = sumOfX + element**2

print("The result is %d" % (sumOfX) )
```

```
1
3
5
7
9
The result is 165
```

Let's solve it together…

4. Write another for-loop to find the sum of the square of the odd elements in x

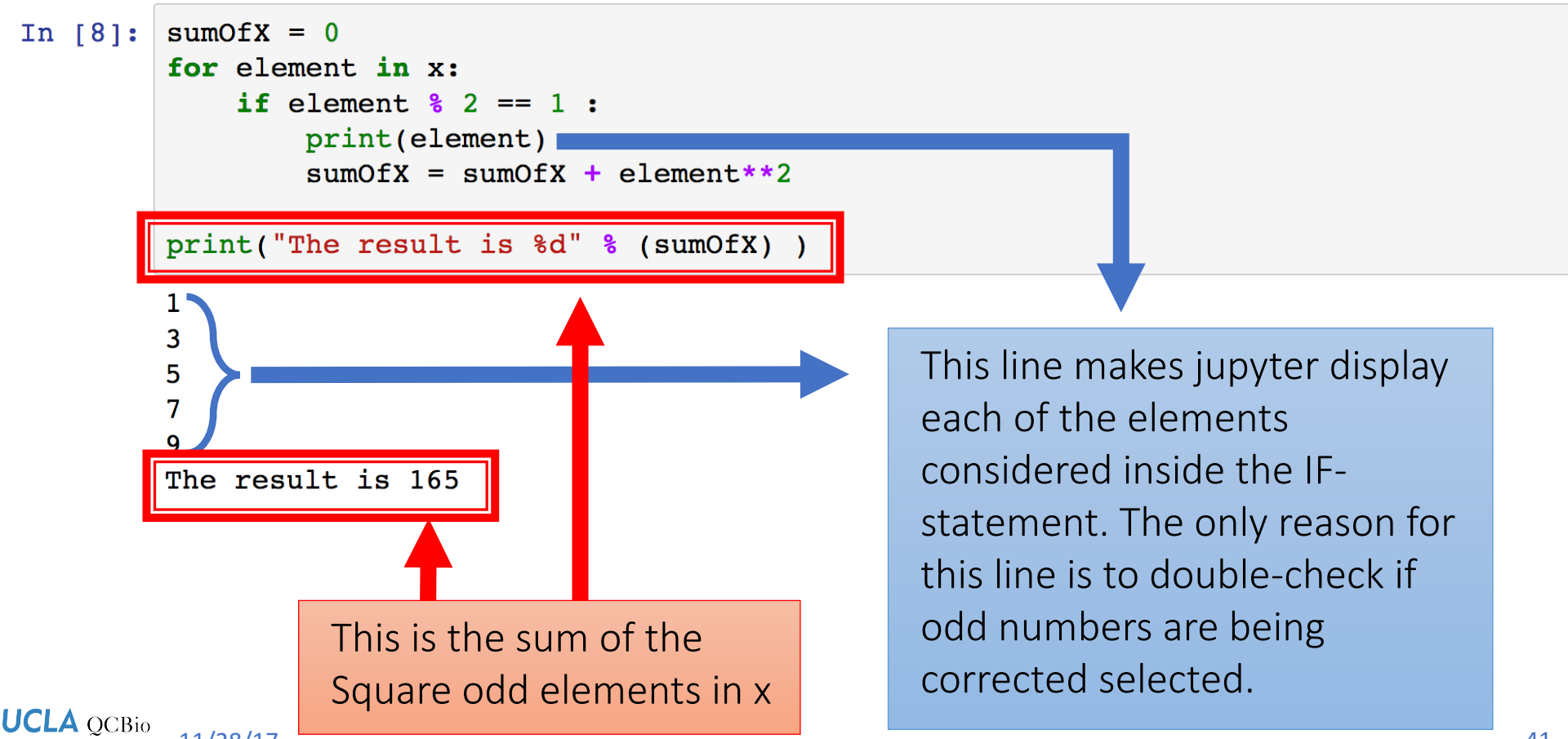
```
In [8]: sumOfX = 0
        for element in x:
            if element % 2 == 1 :
                print(element)
                sumOfX = sumOfX + element**2

        print("The result is %d" % (sumOfX) )
```

```
1
3
5
7
9
The result is 165
```

This line makes jupyter display each of the elements considered inside the IF-statement. The only reason for this line is to double-check if odd numbers are being corrected selected.

Let's solve it together…

4. Write another for-loop to find the sum of the square of the odd elements in x



Execution order

At this point, you should have 7 or more cells

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [3]: 5*3 + 2

Out[3]: 17

In [4]: print(y*z)

        119.6

In [5]: print(x)

        17
```

Execution order

At this point, you should have 7 or more cells

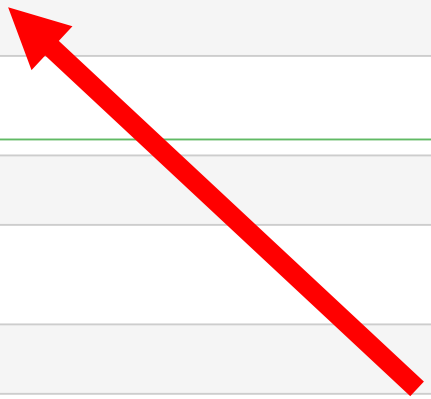
```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [3]: 5*3 + 2

Out[3]: 17

In [4]: print(y*z)

        119.6

In [5]: print(x)

        17
```

Click on this cell

Execution order

At this point, you should have 7 or more cells

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [3]: 5*3 + 2

Out[3]: 17

In [4]: print(y*z)

        119.6

In [5]: print(x)

        17
```



Execute it

Execution order

This time, you should see an error. Can you explain why is this error happening now, but not before?

```
In [9]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-35a6e3b69177> in <module>()
----> 1 y = 7*x**2 + 0.6*x
      2 z = 1/x
      3 y*z

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Execution order

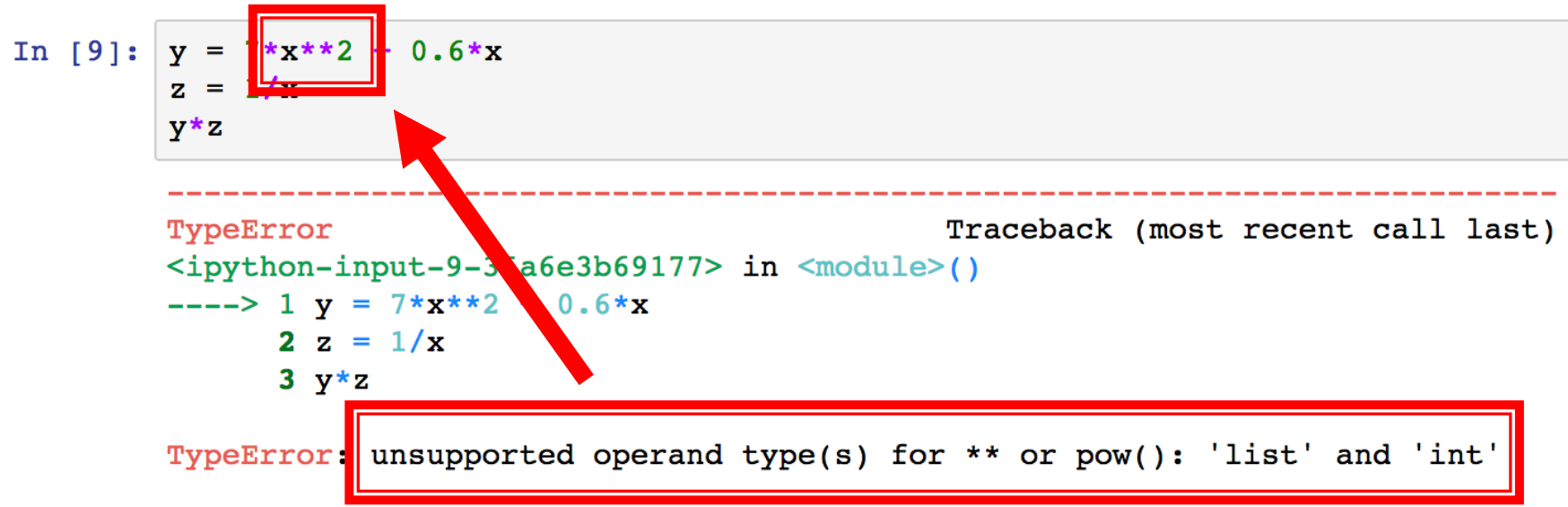
This time, you should see an error. Can you explain why is this error happening now, but not before?

```
In [9]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-35a6e3b69177> in <module>()
----> 1 y = 7*x**2 + 0.6*x
      2 z = 1/x
      3 y*z

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

This gives a very good hint for the reason of this error

Execution order: be careful!

This time, you should see an error. Can you explain why is this error happening now, but not before?



The error message is telling you that the operation ** does not support list and integer

Execution order: be careful!

Here's what happened:

1. The first time, x was an integer

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6
```

2. Then, we re-defined x as a list

```
In [6]: x = [0,1,2,3,4,5,6,7,8,9,10]
        print(x)

        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3. The next time around, the x squared makes no sense because x is a list…

```
In [9]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

---------------------------------------------
TypeError                                 Traceb
<ipython-input-9-35a6e3b69177> in <module>()
----> 1 y = 7*x**2 + 0.6*x
```

Execution order: be careful!

Here's what happened:

1. The first time, x was an integer

```
In [1]: x = 5*3 + 2
```

To fix this, whenever you go back make sure to run all relevant cells!

Try running both 1st and 2nd cells

2. Then

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3. The next time around, the x squared makes no sense because x is a list…

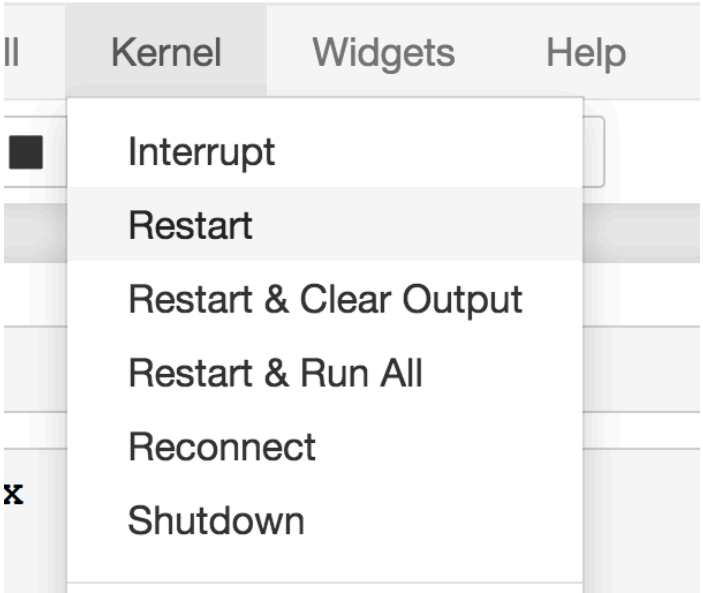
```
In [9]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z
```

```
---------------------------------------------------------
TypeError                                 Traceb
<ipython-input-9-35a6e3b69177> in <module>()
----> 1 y = 7*x**2 + 0.6*x
```

Restarting the notebook

Whenever you are not sure, restart your notebook!

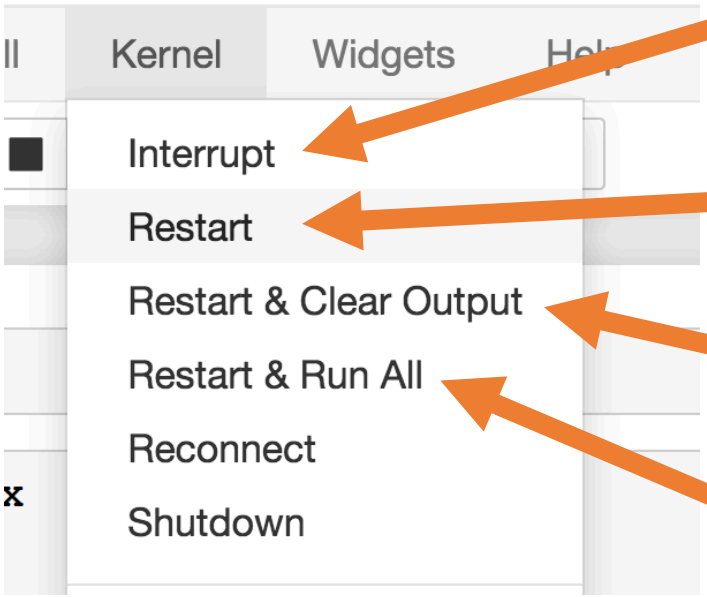
Click on the Kernel button at the top of your notebook:



Resetting the notebook

Whenever you are not sure, reset your notebook!

Click on the Kernel button at the top of your notebook:



Interrupts any cell that may be taking too long to finish running

Restarts everything, clearing the values in all variables

Same as before, but also clears the outputs previously displayed

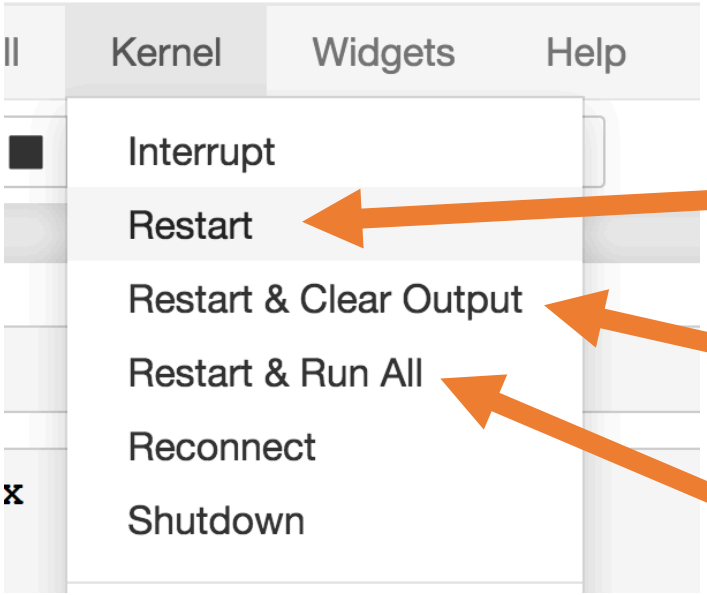
Same as before, but runs every cell after clearing their values

Resetting the notebook

Whenever you are not sure, reset your notebook!

Click on the Kernel button at the top of your notebook:

Try each of the following options below and examine the result of each action on your notebook



Restarts everything, clearing the values in all variables

Same as before, but also clears the outputs previously displayed

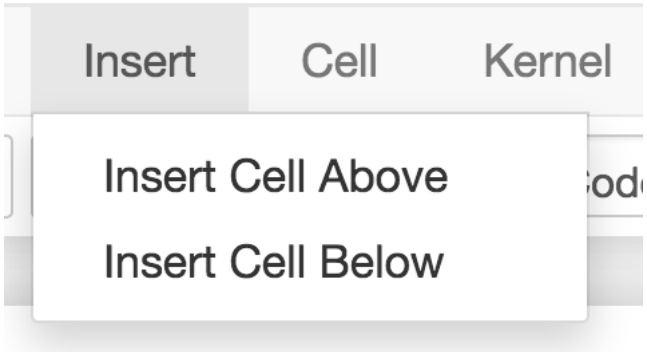
Same as before, but runs every cell after clearing their values

Adding text comments

One last time, click on the second cell:

```
In [1]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6

In [3]: 5*3 + 2

Out[3]: 17
```

Click on Insert and select “Insert Cell Above”



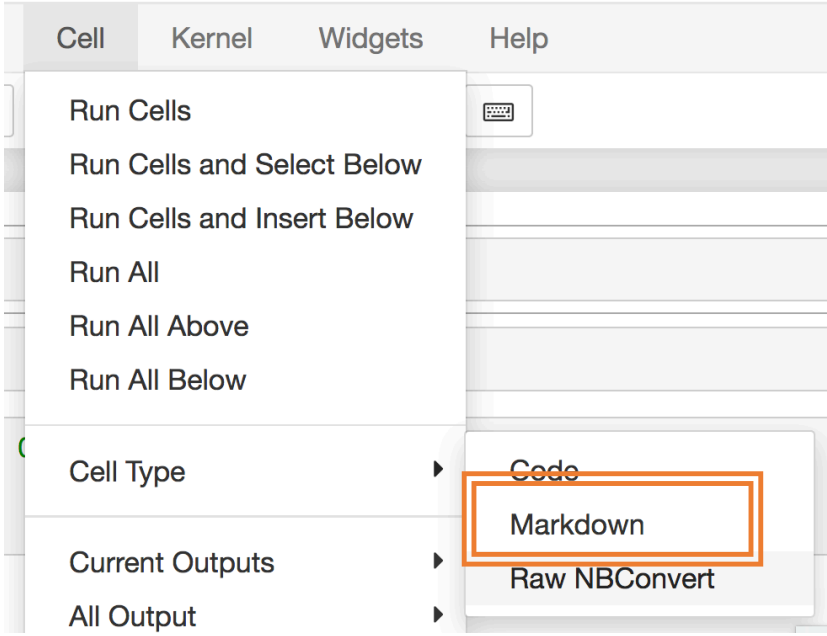
Adding text comments

```
In [1]: x = 5*3 + 2

In [ ]:

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6
```

With the new cell selected, click on the Cell menu at the top.

Go to Cell Type

Select Mark Down



Adding text comments

Note that this cell does not have brackets [] anymore. This cell is now is its "markdown" type and can contain normal text.

```
In [ ]: x = 5*3 + 2

In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z
Out[2]: 119.6
```

Try writing down some text and then execute this cell (by using for instance the same keyboard shortcuts as before).

```
This is some text. We can even use formulas:
$$ \theta = \frac{[L]^n}{K_d + [L]^n} $$
```

Write down anything you want, doesn't need to be the same as above

Adding text comments

Once you execute this cell, this is what you should see:

```
In [ ]: x = 5*3 + 2
```

This is some text. We can even use formulas:

$$\theta = \frac{[L]^n}{K_d + [L]^n}$$

```
In [2]: y = 7*x**2 + 0.6*x
        z = 1/x
        y*z

Out[2]: 119.6
```

Jupyter Notebooks become especially powerful when you start making use of both coding and markdown cells

Example: writing reports and sharing with collaborators

Saving and loading notebooks

To save your notebook, you can use the following keyboard shortcut:

MAC:	Command + s
WINDOWS:	Control + s

By default, notebooks are saved at the same folder your created (or loaded) your notebook (from).

Saving and loading notebooks

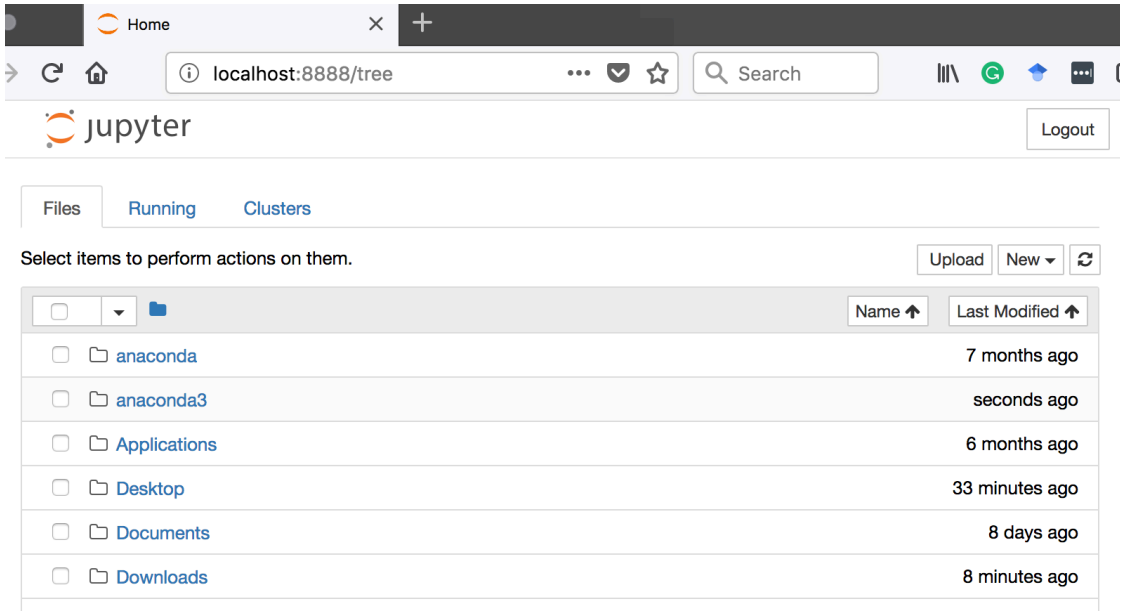
To save your notebook, you can use the following keyboard shortcut:

MAC: Command + s

WINDOWS: Control + s

By default, notebooks are saved at the same folder your created (or loaded) your notebook (from).

Try now navigating to the same folder you created your notebook and look for a file with extension .ipynb

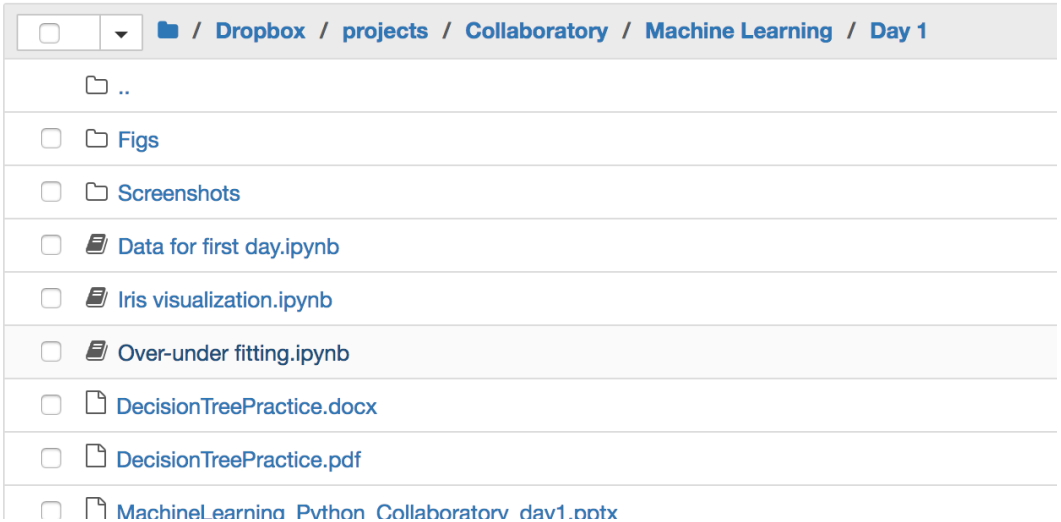


Saving and loading notebooks

Notebooks are files as any other, and can be shared, copied and moved to another location

To open a notebook, simply navigate using the Jupyter Dashboard and click on the notebook you want o load

Notebooks have a small gray icon



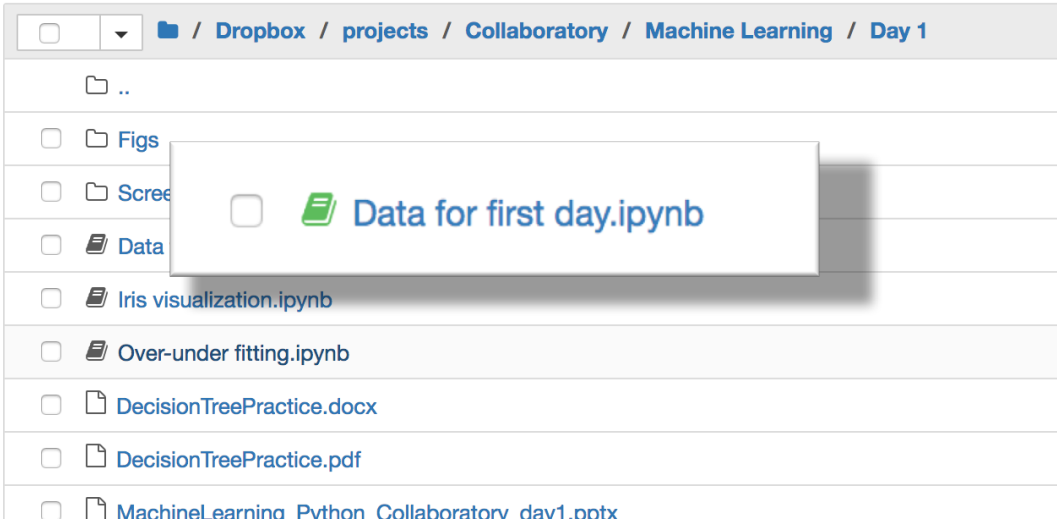
Saving and loading notebooks

Notebooks are files as any other, and can be shared, copied and moved to another location

To open a notebook, simply navigate using the Jupyter Dashboard and click on the notebook you want o load

Notebooks have a small gray icon

If the icon is green, then that notebook is still active and may be open in another tab

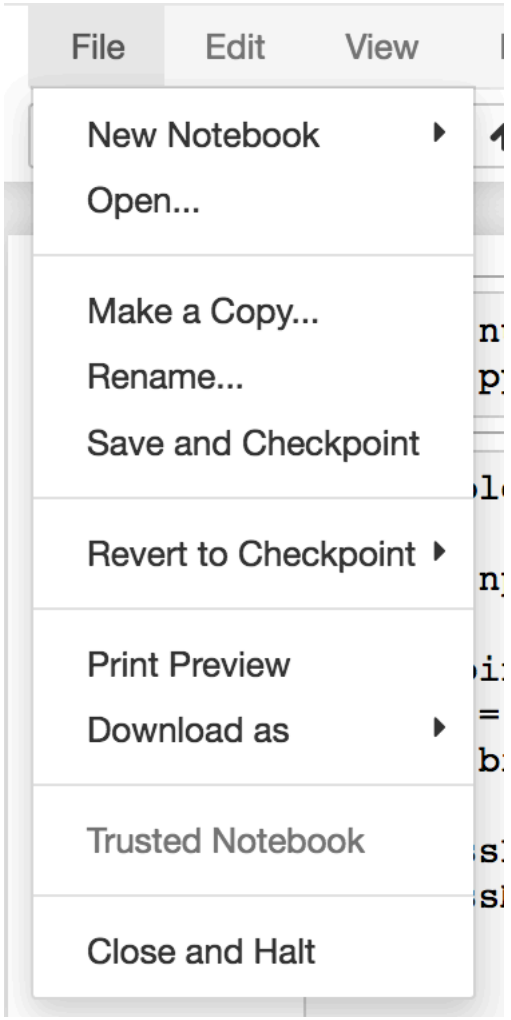


Properly closing a notebook

After saving it, to close a notebook go to File and click on Close and Halt

We encourage you to always make sure the notebook was properly halted before ending your Jupyter session.

Check on your Jupyter Dashboard if the notebook icon is not green anymore. That means your notebook was closed successfully.



This is all for now…

- We will use Jupyter Notebooks throughout this workshops, and will cover more tricks and useful features
- However, the best way to learn how to leverage the full potential of notebooks is by using them!
- Jupyter Notebooks are famous for **not** requiring a steep learning curve and providing a visual and interactive connection to Python and other programming languages
- Don’t hesitate asking questions at any given time!

Examples of cool notebooks

Here are a few notebooks that may help you find inspirations for your own notebooks.

If you see something you like, download it and examine how they did it!

For instance:

http://nbviewer.jupyter.org/github/thmosqueiro/modeligem/blob/master/notebooks/Data_analysis.ipynb

For more examples of notebooks, go to

http://thmosqueiro.vandroiy.com/MachineLearning_CollW

Some important libraries for Machine Learning

Libraries in Python

Libraries are an important characteristic of how Python works, and some would say it is the reason why Python is so popular

NumPy

NumPy is a library for large multi-dimensional arrays and matrices, and math operations with them.

Numpy is widely used by many other libraries under the hood, including Scikit-Learn.

Let's continue the same notebook to explore some of the features offered by Numpy

To use numpy in your code, simply use the import statement in the next cell:

```
In [9]: import numpy
```

Numpy – Creating arrays

NumPy arrays provide the usual matrix and vector operations

To create a NumPy array, use:

```
Variable = numpy.array( Array_Like )
```

Example:

```
In [9]: import numpy

In [10]: a = [1,2,3,4,5]
         print( type(a) )   # displays the type of the variable a

         b = numpy.array( [1,2,3,4,5] )
         print( type(b) )   # displays the type of the variable b

         <class 'list'>
         <class 'numpy.ndarray'>
```

Numpy – Creating arrays

NumPy arrays provide the usual matrix and vector operations

To create a NumPy array, use:

```
Variable = numpy.array( Array_Like )
```

Example:

```
In [9]: import numpy

In [10]: a = [1,2,3,4,5]
         print( type(a) )   # displays the type of the variable a

         b = numpy.array( [1,2,3,4,5] )
         print( type(b) )   # displays the type of the variable b

         <class 'list'>
         <class 'numpy.ndarray'>
```

Numpy arrays

It is common to rename NumPy as np:

In [11]:
```
import numpy as np  # allows user to use np instead of numpy

c = np.array( [5,4,3,2,1] )
print( type(c) )    # displays the type of the variable c
```

```
<class 'numpy.ndarray'>
```

Mathematical operations with NumPy arrays:

In [12]:
```
print("Sum of lists: ", a + a)

print("Sum of NumPy arrays: ", b + c )
```

```
Sum of lists:  [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
Sum of NumPy arrays:  [6 6 6 6 6]
```

Summing two NumPy arrays is indeed the element-wise summation

Mathematical operators

Here is how NumPy arrays behave

```
In [13]: print("Original array: ", b)
         print("Summing a constant: ", b + 7.1 )
         print("Multiplying by a constant: ", b*2.5 )
         print("Power function: ", b**3 )
         print("Element-wise product: ", b*c )
         print("Element-wise division: ", b/c )
         print("Element-wise modulo: ", b%c )

Original array:  [1 2 3 4 5]
Summing a constant:  [  8.1   9.1  10.1  11.1  12.1]
Multiplying by a constant:  [  2.5   5.    7.5  10.   12.5]
Power function:  [  1   8  27  64 125]
Element-wise product:  [5 8 9 8 5]
Element-wise division:  [ 0.2  0.5  1.   2.   5. ]
Element-wise modulo:  [1 2 0 0 0]
```

Other functions and indexing

Some mathematical functions:

```
In [14]: print("Exponential: ", np.exp(b) )
         print("Sine: ", np.sin(b) )
         print("Cosine: ", np.cos(b) )
```

```
Exponential:  [   2.71828183    7.3890561    20.08553692   54.59815003  148.4131591 ]
Sine:  [ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
Cosine:  [ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
```

You can also select specific elements:

```
In [17]: print("First element in b: ", b[0] )
         print("Second element in b: ", b[1] )
         print("Last element in b: ", b[-1] )
         print("All elements in b: ", b[:] )
         print("Elements from 1 to 3: ", b[1:4] ) # last one doesn't count!
         print("From 0 to end in steps of 2: ", b[0:-1:2] )
```

```
First element in b:  1
Second element in b:  2
Last element in b:  5
All elements in b:  [1 2 3 4 5]
Elements from 1 to 3:  [2 3 4]
From 0 to end in steps of 2:  [1 3]
```

Other functions and indexing

Some mathematical functions:

```
In [14]: print("Exponential: ", np.exp(b) )
         print("Sine: ", np.sin(b) )
         print("Cosine: ", np.cos(b) )
```

```
Exponential:  [   2.71828183    7.3890561                                   31591 ]
Sine:  [ 0.84147098  0.90929743  0.1411200
Cosine:  [ 0.54030231 -0.41614684 -0.9899
```

The syntax is always:

START:END:STEP

- if step=1, you can omit step
  eg. b[1:4]
- if end=-1 you can omit end
  eg. b[3:]
- if start=0, you can omit start
  eg. b[:3]

You can also select specific elemen

```
In [17]: print("First element in b: ", b[0] )
         print("Second element in b: ", b[1] )
         print("Last element in b: ", b[-1] )
         print("All elements in b: ", b[:] )
         print("Elements from 1 to 3: ", b[1:4] ) # last one doesn't count!
         print("From 0 to end in steps of 2: ", b[0:-1:2] )
```

```
First element in b:  1
Second element in b:  2
Last element in b:  5
All elements in b:  [1 2 3 4 5]
Elements from 1 to 3:  [2 3 4]
From 0 to end in steps of 2:  [1 3]
```

Numpy arrays as… "objects"

NumPy arrays (as any other variable) carry a baggage with them: we call them attributes and methods

Attributes are "properties" of an object

E.g. arrays have a shape, which tells you their size in each dimension

```
In [19]: b.shape
Out[19]: (5,)
```

Methods are "actions" (or functions) applied to or performed on the object

E.g. sum() performs the sum of all values of elements in an array

```
In [20]: b.sum()
Out[20]: 15
```

Numpy arrays as… "objects"

NumPy arrays (as any other variable) carry a baggage with them: we call them attributes and methods

Attributes are "properties" of an object

E.g. arrays have a shape, which tells you t

```
In [19]: b.shape
Out[19]: (5,)
```

Methods are "actions" (or function on the object

E.g. sum() performs the sum of all valu

```
In [20]: b.sum()
Out[20]: 15
```

Both attributes and methods are accessed or executed by using the . (dot) after the object

Methods use parenthesis () because they work just like functions. Attributes don't.

Finding methods & attributes

Python's documentation is written by a large community, with diverse background and points of view

We strongly encourage you to learn to use it

All of the information is freely available to anyone, and will be always kept free (as in "free will")

Let's give it a try:

Open your browser and look for

"numpy array attributes"

numpy array attributes

Web Images Videos

Brazil Safe Search: Strict Any Time

The N-dimensional **array** (ndarray) — **NumPy** v1.13 Manual - SciPy

The N-dimensional **array** (ndarray) ... N integers), and via the methods and **attributes** of the ndarray. ... The Scipy community.

https://docs.scipy.org/doc/numpy/reference/arrays.nd... More results

NumPy - **Array Attributes** - SAP Hybris, FlexBox, Axure RP ...

NumPy Array Attributes - Learn **NumPy** in simple and easy steps starting from basic to advanced concepts with examples including Introduction, Environment, Ndarray ...

https://www.tutorialspoint.com/numpy/numpy_array_attributes.htm

python - **Numpy array** of object **attributes** - Stack Overflow

I have a multi-dimensional **array** of objects, something like: a = np.array([obj1,obj2,obj3]) The objects are instances of a class which has several **attributes**. Let's ...

https://stackoverflow.com/questions/18577013/numpy-array-of-object-...

Python **attributes** and **numpy arrays** - Stack Overflow

I have a class that stores some **attributes**. These **attributes** are **numpy arrays** with some floats inside. I want this **attributes** to be accessed when creating objects.

https://stackoverflow.com/questions/16941487/python-attributes-and-...

Docs on NumPy Arrays

SciPy.org Sponsored By ENTHOUGHT

The N-dimensional array (`ndarray`)

An `ndarray` is a (usually fixed-size) multidimensional container of items of the same type and size. The number of dimensions and items in an array is defined by its `shape`, which is a `tuple` of *N* positive integers that specify the sizes of each dimension. The type of items in the array is specified by a separate data-type object (dtype), one of which is associated with each ndarray.

As with other container objects in Python, the contents of an `ndarray` can be accessed and modified by indexing or slicing the array (using, for example, *N* integers), and via the methods and attributes of the `ndarray`.

Different `ndarrays` can share the same data, so that changes made in one `ndarray` may be visible in another. That is, an ndarray can be a *"view"* to another ndarray, and the data it is referring to is taken care of by the *"base"* ndarray. ndarrays can also be views to memory owned by Python `strings` or objects implementing the `buffer` or array interfaces.

Example:
A 2-dimensional array of size 2 x 3, composed of 4-byte integer elements:

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
>>> type(x)
<type 'numpy.ndarray'>
>>> x.shape
(2, 3)
>>> x.dtype
dtype('int32')
```

The array can be indexed using Python container-like syntax:

Table Of Contents

Previous topic

Docs on NumPy Arrays

In the documentation, look for Array attributes

Array attributes

Array attributes reflect information that is intrinsic to the array itself. Generally, accessing an array through its attributes allows you to get and sometimes set intrinsic properties of the array without creating a new array. The exposed attributes are the core parts of an array and only some of them can be reset meaningfully without creating a new array. Information on each attribute is given below.

Memory layout

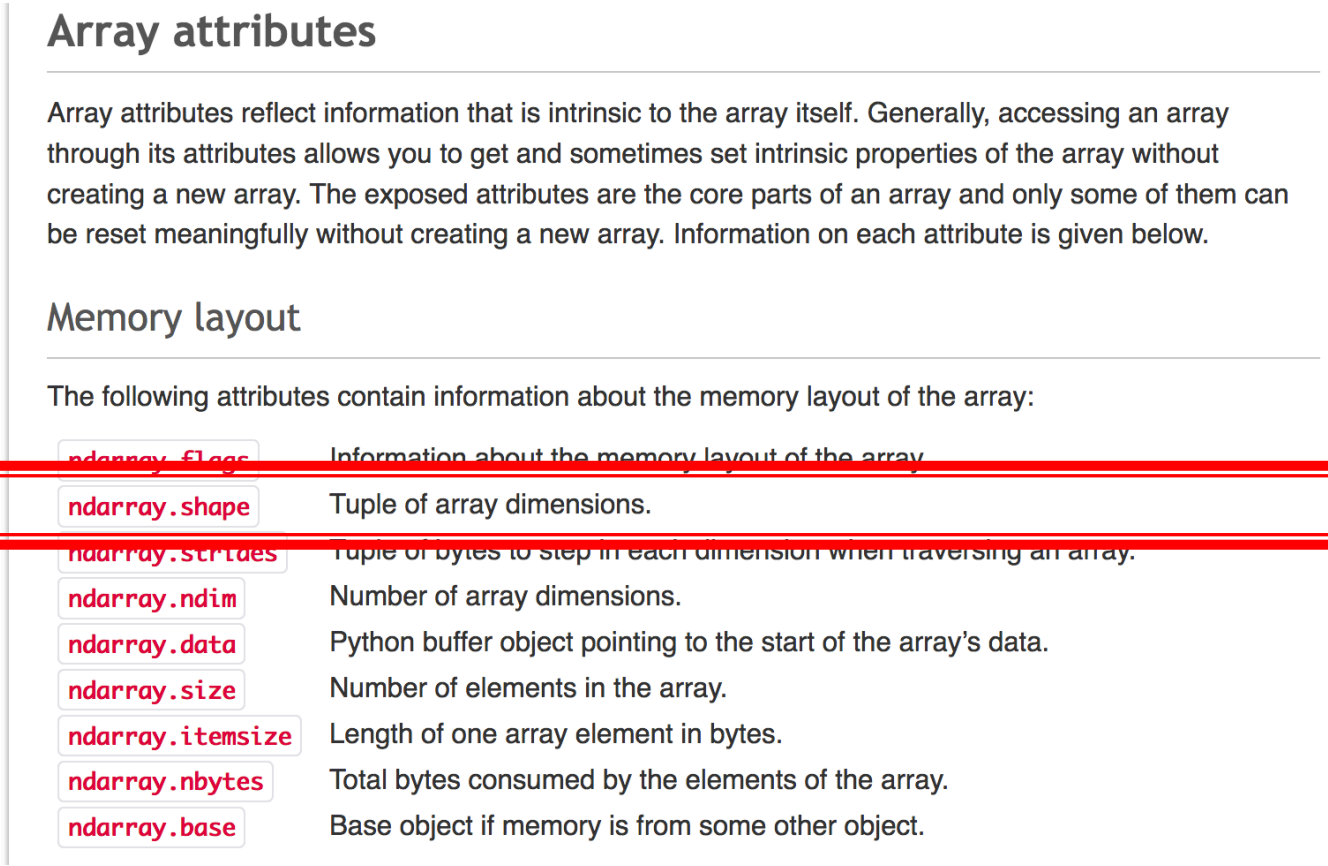
The following attributes contain information about the memory layout of the array:

`ndarray.flags`	Information about the memory layout of the array.
`ndarray.shape`	Tuple of array dimensions.
`ndarray.strides`	Tuple of bytes to step in each dimension when traversing an array.
`ndarray.ndim`	Number of array dimensions.
`ndarray.data`	Python buffer object pointing to the start of the array's data.
`ndarray.size`	Number of elements in the array.
`ndarray.itemsize`	Length of one array element in bytes.
`ndarray.nbytes`	Total bytes consumed by the elements of the array.
`ndarray.base`	Base object if memory is from some other object.

Try for yourself the attributes size and ndim

Docs on NumPy Arrays

In the documentation, look for Array attributes



Array attributes

Array attributes reflect information that is intrinsic to the array itself. Generally, accessing an array through its attributes allows you to get and sometimes set intrinsic properties of the array without creating a new array. The exposed attributes are the core parts of an array and only some of them can be reset meaningfully without creating a new array. Information on each attribute is given below.

Memory layout

The following attributes contain information about the memory layout of the array:

`ndarray.flags`	Information about the memory layout of the array.
`ndarray.shape`	Tuple of array dimensions.
`ndarray.strides`	Tuple of bytes to step in each dimension when traversing an array.
`ndarray.ndim`	Number of array dimensions.
`ndarray.data`	Python buffer object pointing to the start of the array's data.
`ndarray.size`	Number of elements in the array.
`ndarray.itemsize`	Length of one array element in bytes.
`ndarray.nbytes`	Total bytes consumed by the elements of the array.
`ndarray.base`	Base object if memory is from some other object.

Try for yourself the attributes size and ndim

Docs on NumPy Arrays

Now, look for Array methods

> ## Array methods
>
> An `ndarray` object has many methods which operate on or with the array in some fashion, typically returning an array result. These methods are briefly explained below. (Each method's docstring has a more complete description.)
>
> For the following methods there are also corresponding functions in `numpy`: `all`, `any`, `argmax`, `argmin`, `argpartition`, `argsort`, `choose`, `clip`, `compress`, `copy`, `cumprod`, `cumsum`, `diagonal`, `imag`, `max`, `mean`, `min`, `nonzero`, `partition`, `prod`, `ptp`, `put`, `ravel`, `real`, `repeat`, `reshape`, `round`, `searchsorted`, `sort`, `squeeze`, `std`, `sum`, `swapaxes`, `take`, `trace`, `transpose`, `var`.

Click on the method sum() to see more information

Try using for yourself the following methods: cumsum, mean(), max() and min()

Methods and attributes

```
In [21]: print("Size attribute: ", b.size)
         print("ndim attribute: ", b.ndim)
```

```
Size attribute:  5
ndim attribute:  1
```

```
In [22]: print("Cumulative sum: ", b.cumsum())
         print("Mean of elements: ", b.mean())
         print("Max of elements: ", b.max())
         print("Min of elements: ", b.min())
```

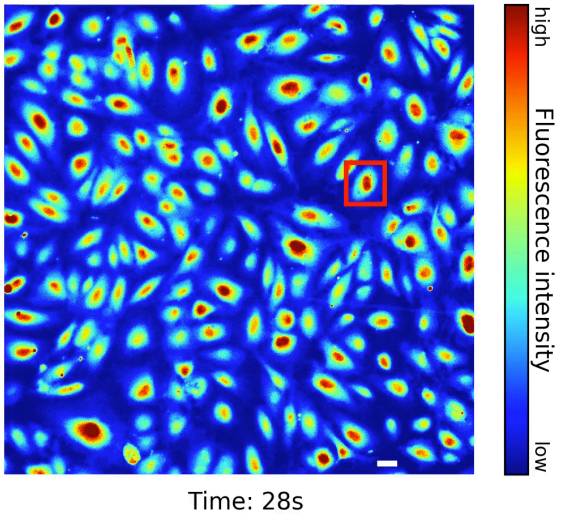
```
Cumulative sum:  [ 1  3  6 10 15]
Mean of elements:  3.0
Max of elements:  5
Min of elements:  1
```

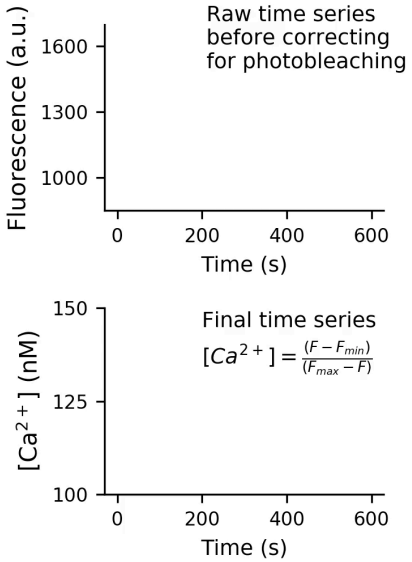
It is important to stress that learning all there is to know about this topic is far from our goal here!
You can most surely learn this as you practice, but if you further material go to our website and look for “More about Object Orientation”!

How about plotting in Python?

The primary and most used plotting library is Python is called Matplotlib, and allows the creation of publication-quality plots and animations.

Example from J Mack et al, Nature Communications v8 p1620 (2017)
https://www.nature.com/articles/s41467-017-01741-8#Sec32



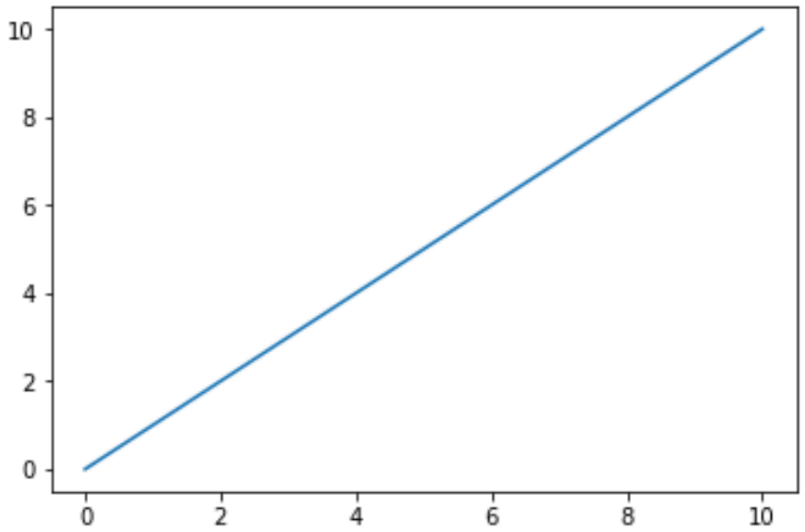


We will be using something called pylab, which is part of matplotlib:

```
In [24]: import pylab as pl

         pl.plot(x)
         pl.show()
```

Simple plotting

```
In [24]: import pylab as pl

         pl.plot(x)
         pl.show()
```



`pl` or `plt` are common shorter names for pylab

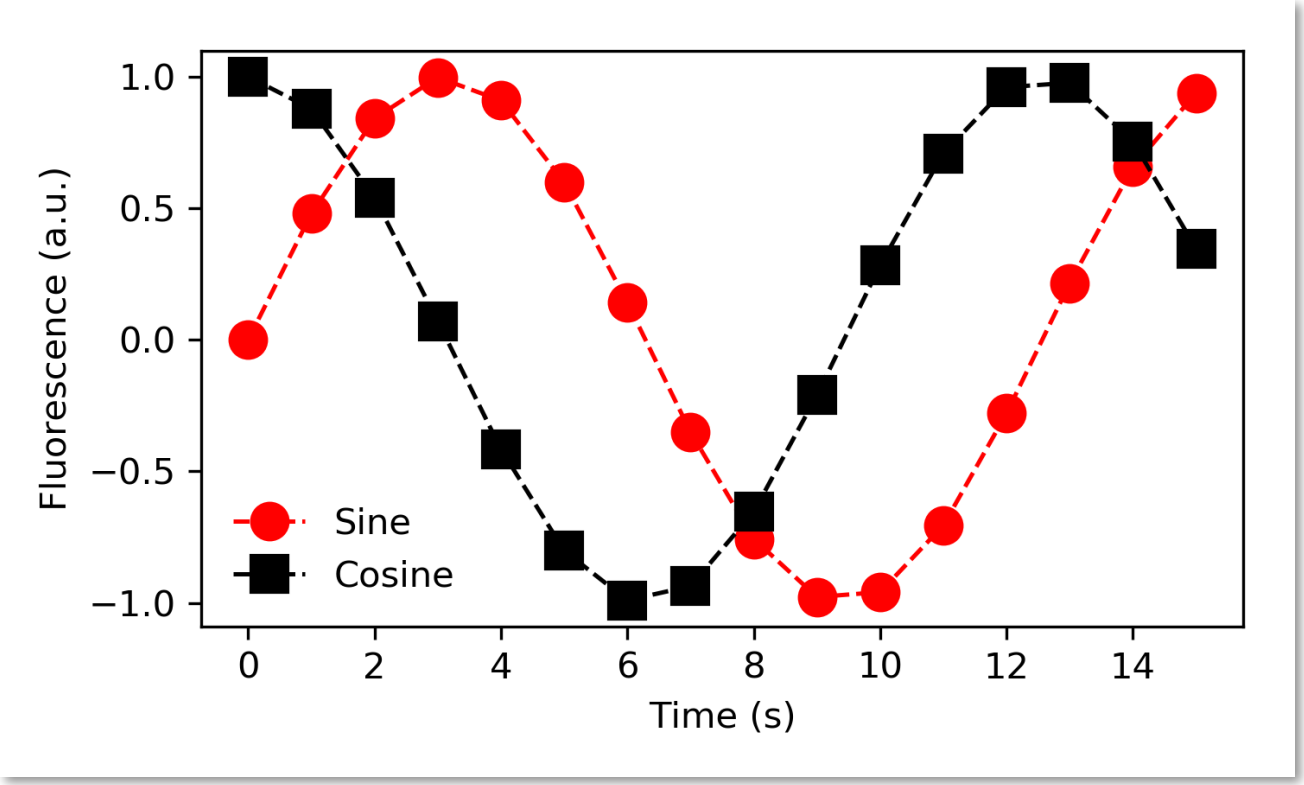
`pl.plot(x)` assigns values in x to a y-axis

`pl.show()` displays the resulting plot

Let's explore pylab by examining an example of plot and then understanding how it was created

Example of a complete plot

Let's construct the following plot:



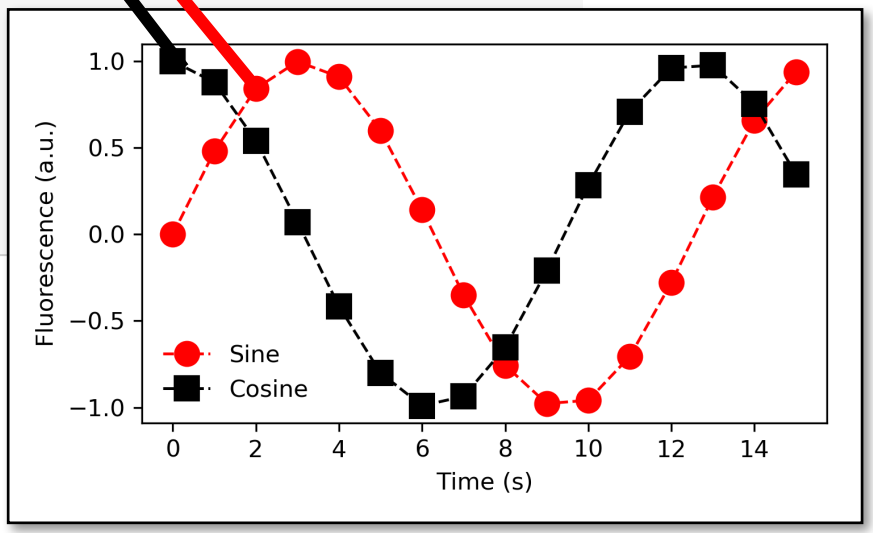
Example of a complete plot

In [24]:
```
x = np.array( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] )
y1 = np.sin(0.5*x)
y2 = np.cos(0.5*x)

pl.plot(x, y1, 'o--', markersize=10, linewidth=1.2,
            color='r', label='Sine')
pl.plot(x, y2, 's--', markersize=10, linewidth=1.2,
            color='k', label='Cosine')

pl.xlabel('Time (s)')
pl.ylabel('Fluorescence (a.u.)')

pl.yticks([-1,-0.5,0.0,0.5,1.0])

pl.legend( frameon=False )
pl.show()
```

Example of a complete plot

```
In [24]: x = np.array( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] )
         y1 = np.sin(0.5*x)
         y2 = np.cos(0.5*x)

         pl.plot(x, y1, 'o--', markersize=10, linewidth=1.2,
                    color='r', label='Sine')
         pl.plot(x, y2, 's--', markersize=10, linewidth=1.2,
                    color='k', label='Cosine')

         pl.xlabel('Time (s)')
         pl.ylabel('Fluorescence (a.u.)')

         pl.yticks([-1,-0.5,0.0,0.5,1.0])

         pl.legend( frameon=False )
         pl.show()
```

Until you use a pl.show(), all pl.plot()'s are put together in the same plot



Example of a complete plot

```
In [24]: x = np.array( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] )
         y1 = np.sin(0.5*x)
         y2 = np.cos(0.5*x)

         pl.plot(x, y1, 'o--', markersize=10, linewidth=1.2,
                    color='r', label='Sine')
         pl.plot(x, y2, 's--', markersize=10, linewidth=1.2,
                    color='k', label='Cosine')

         pl.xlabel('Time (s)')
         pl.ylabel('Fluorescence (a.u.)')

         pl.yticks([-1,-0.5,0.0,0.5,1.0])

         pl.legend( frameon=False )
         pl.show()
```

Creating arrays to plot

Two distinct plots

Setting axes labels

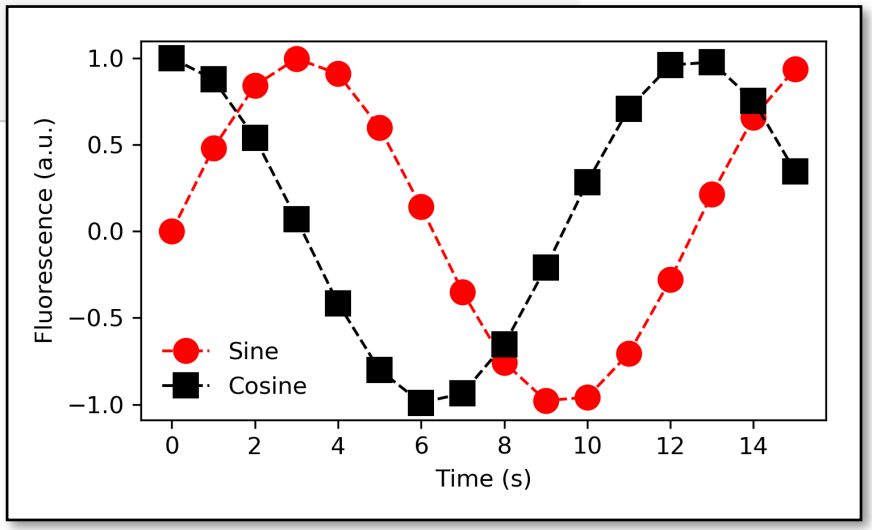
Example of a complete plot

```
In [24]: x = np.array( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] )
         y1 = np.sin(0.5*x)
         y2 = np.cos(0.5*x)

         pl.plot(x, y1, 'o--', markersize=10, linewidth=1.2,
                    color='r', label='Sine')
         pl.plot(x, y2, 's--', markersize=10, linewidth=1.2,
                    color='k', label='Cosine')

         pl.xlabel('Time (s)')
         pl.ylabel('Fluorescence (a.u.)')

         pl.yticks([-1,-0.5,0.0,0.5,1.0])

         pl.legend( frameon=False )
         pl.show()
```

Creating arrays to plot

Two distinct plots

Setting axes labels



Example of a complete plot

```
In [24]: x = np.array( [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] )
         y1 = np.sin(0.5*x)
         y2 = np.cos(0.5*x)

         pl.plot(x, y1, 'o--', markersize=10, linewidth=1.2,
                    color='r', label='Sine')
         pl.plot(x, y2, 's--', markersize=10, linewidth=1.2,
                    color='k', label='Cosine')

         pl.xlabel('Time (s)')
         pl.ylabel('Fluorescence (a.u.)')

         pl.yticks([-1,-0.5,0.0,0.5,1.0])

         pl.legend( frameon=False )
         pl.show()
```

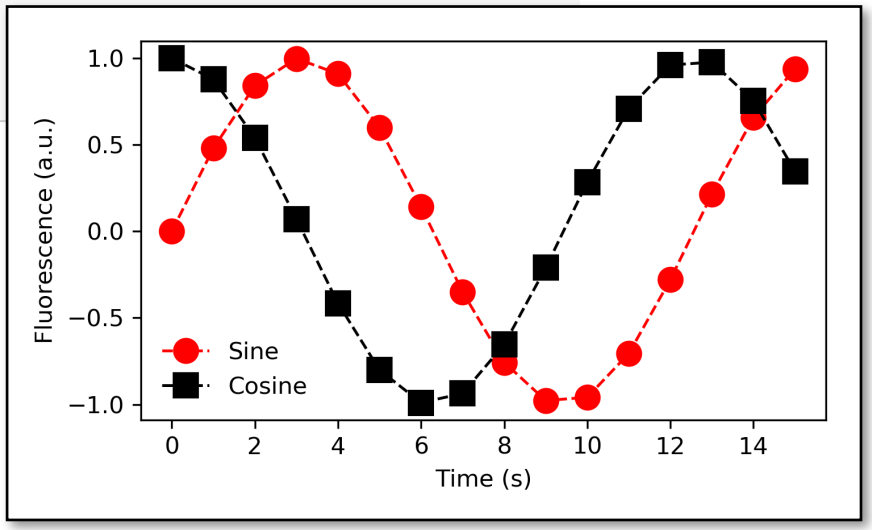
Creating arrays to plot

Two distinct plots

Setting axes labels

Drawing a legend

Selecting where ticks appear in the y-axis



Saving your plot

If you want to save your plot, use pl.savefig() instead of pl.show()

```
f = pl.figure( figsize=(5,3) )

...
plotting...
...

pl.tight_layout()
pl.savefig('Fig1.png', dpi=300)
```

Saving your plot

If you want to save your plot, use pl.savefig() instead of pl.show()

```
f = pl.figure( figsize=(5,3) )

...
plotting...
...

pl.tight_layout()
pl.savefig('Fig1.png', dpi=300)
```

This defines the length of each dimension of your figure (in inches)

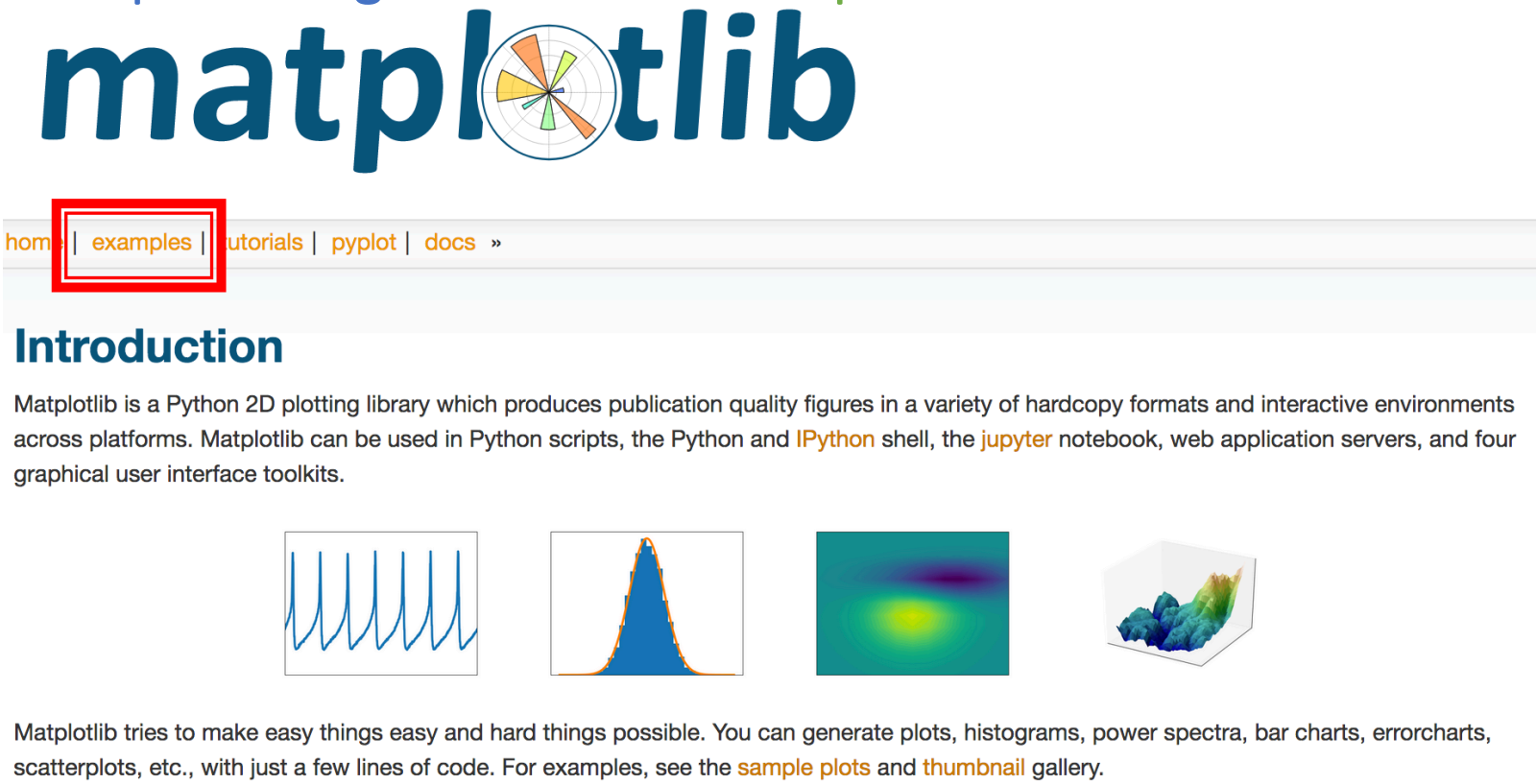
pl.tight_layout() fixes margins and the spacing among the plot elements

This saved the figure as a PNG file, using 300 dots per inch

How do I plot…..

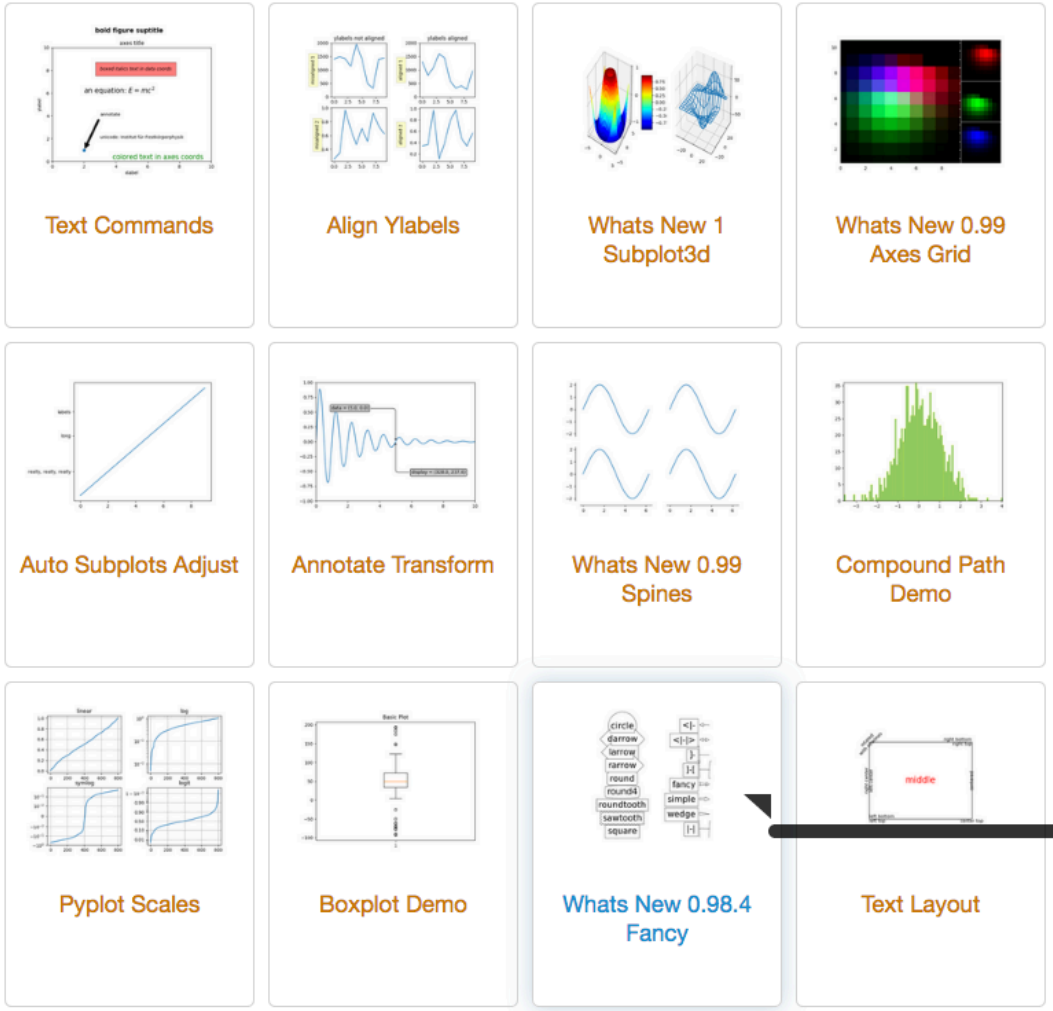
Maplotlib has extensive documentation, but it also has a very handy gallery of examples

Go to matplotlib.org and click on examples



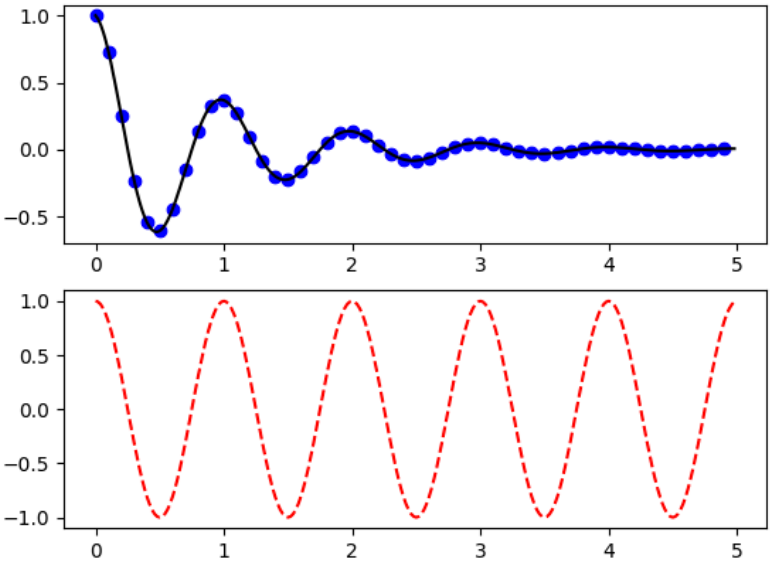
How do I plot……

Select a type of plot you would like to learn to reproduce with your own data and select it



How do I plot…..

Pyplot Two Subplots




```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Below an example of how the plot will look like, you will see the code that generated it.

This is how most people learn to use matplotlib

NumPy and Matplotlib

Throughout the rest of this workshop, we will make extensive use of these two libraries

Nowadays, these are the flagship in many projects not only in academia, but in the industry of Data Science

However, especially in a first contact, going through all details about in these libraries is not the most productive approach

With this introduction to both libraries, you will be able to get started and learn more from there

Overview of Machine Learning

Types of Learning

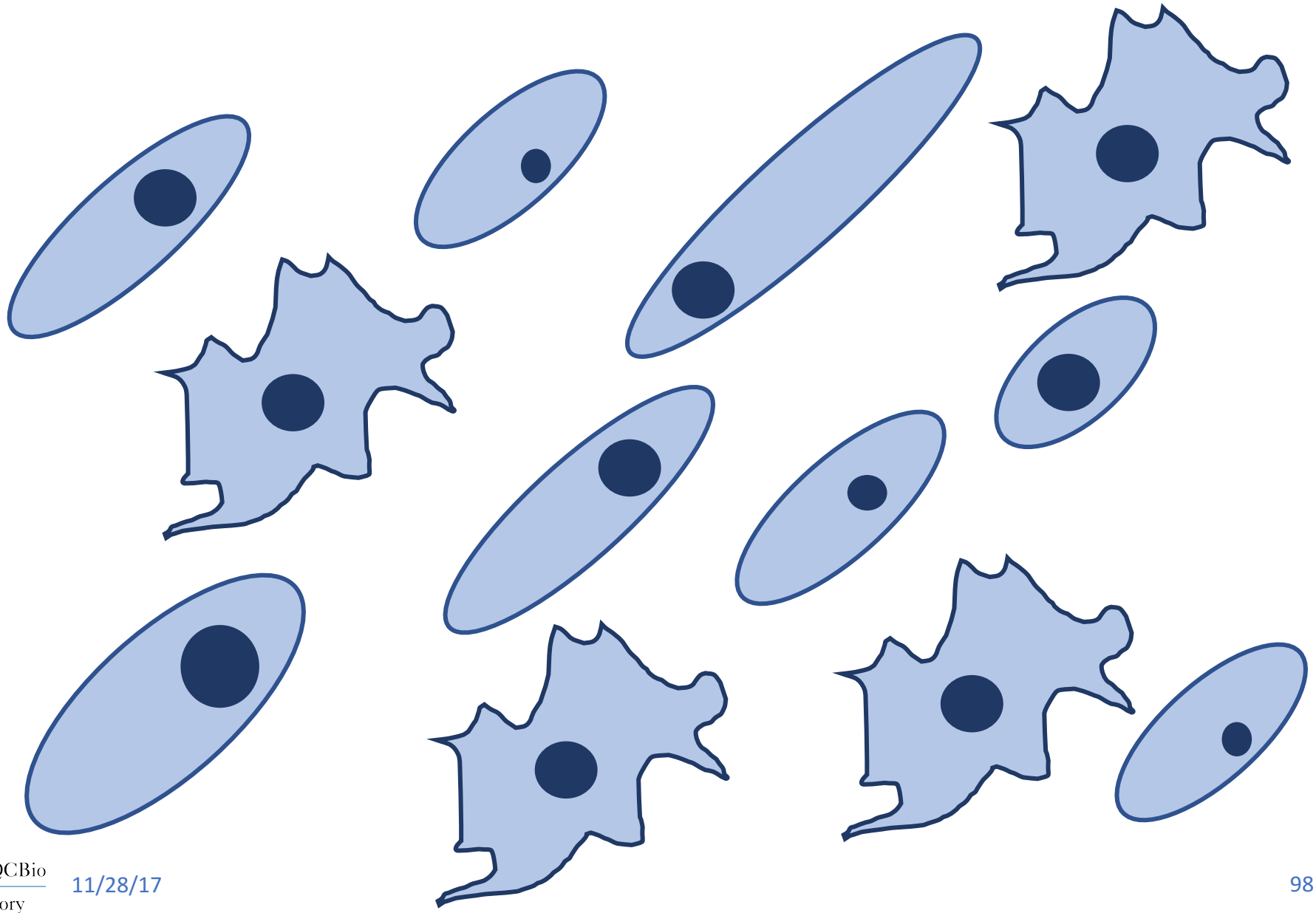
Machine learning is classically divided into three domains:

1. Supervised learning
   - Classification
   - Regression
2. Unsupervised learning
3. Semi-supervised learning

   …

In this workshop, we heavily focus on the first 1 categories. We will touch Unsupervised Learning on Day 3.

Types of Supervised Learning



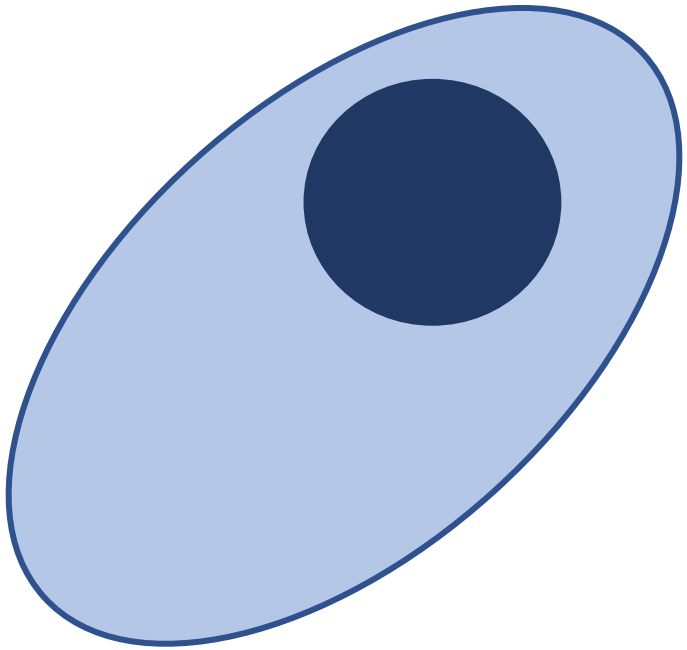
Target and features

In this scenario you could…

1. Create a model that identifies cells by their shape
2. Create a function that estimates the cell volume

The first step is select features

Features are individual measurable properties of the object (or phenomenon) being studies



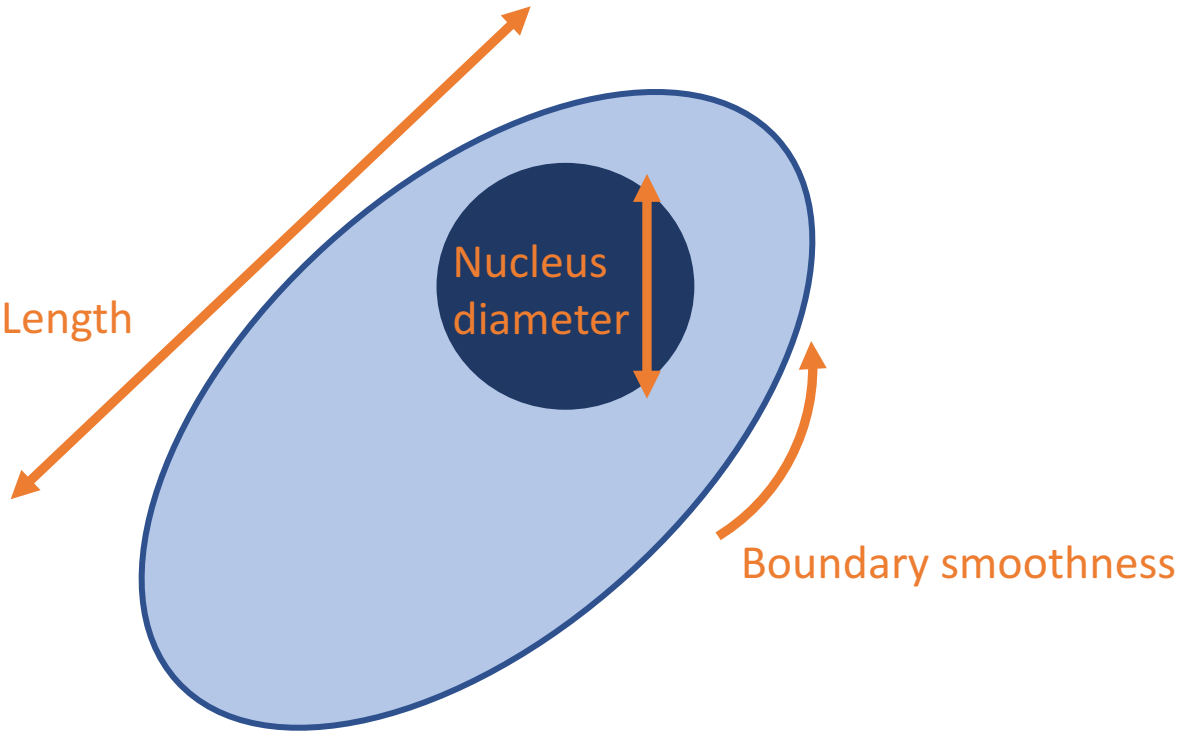
Target and features

In this scenario you could…

1. Create a model that identifies cells by their shape
2. Create a function that estimates the cell volume

The first step is select features

Features are individual measurable properties of the object (or phenomenon) being studies



Target and features

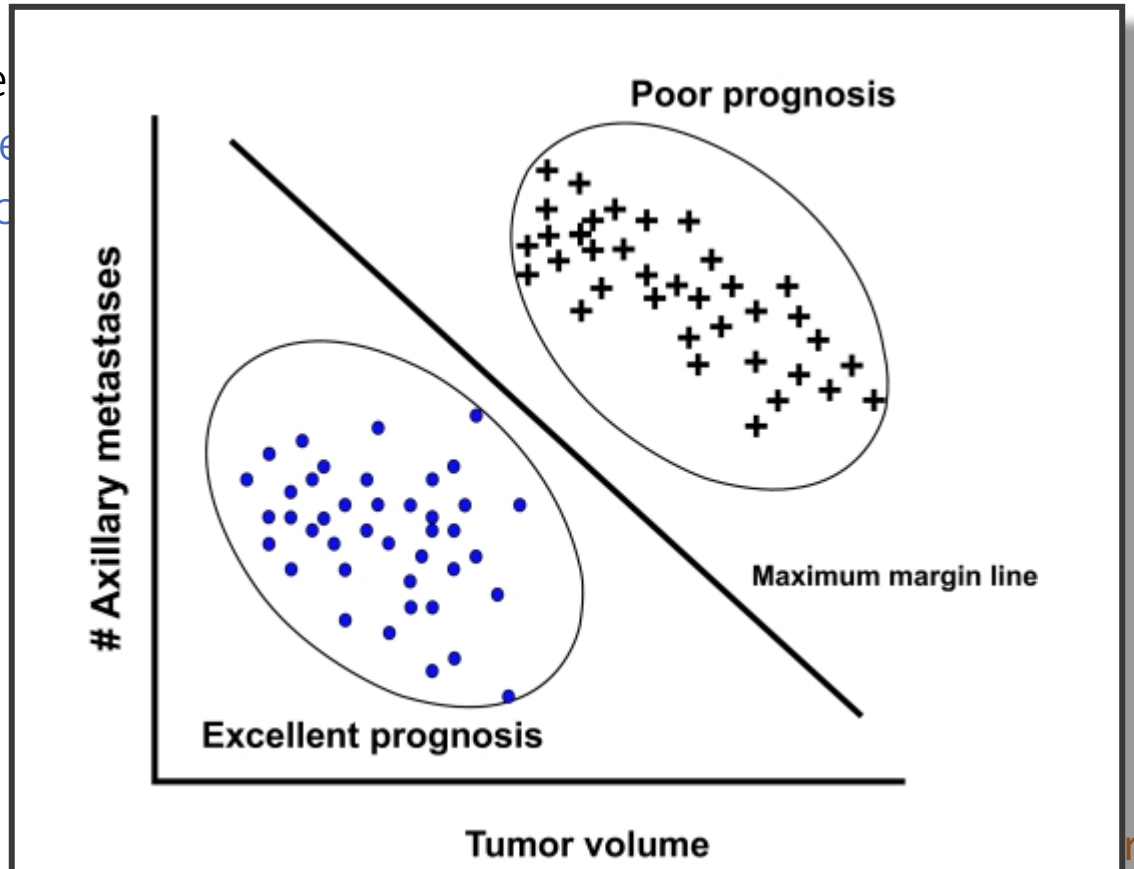
In this scenario you could…

1. Create a model that identifies cells by their shape
2. Create a function that estimates the cell volume

The first ste

Features are

phenomeno

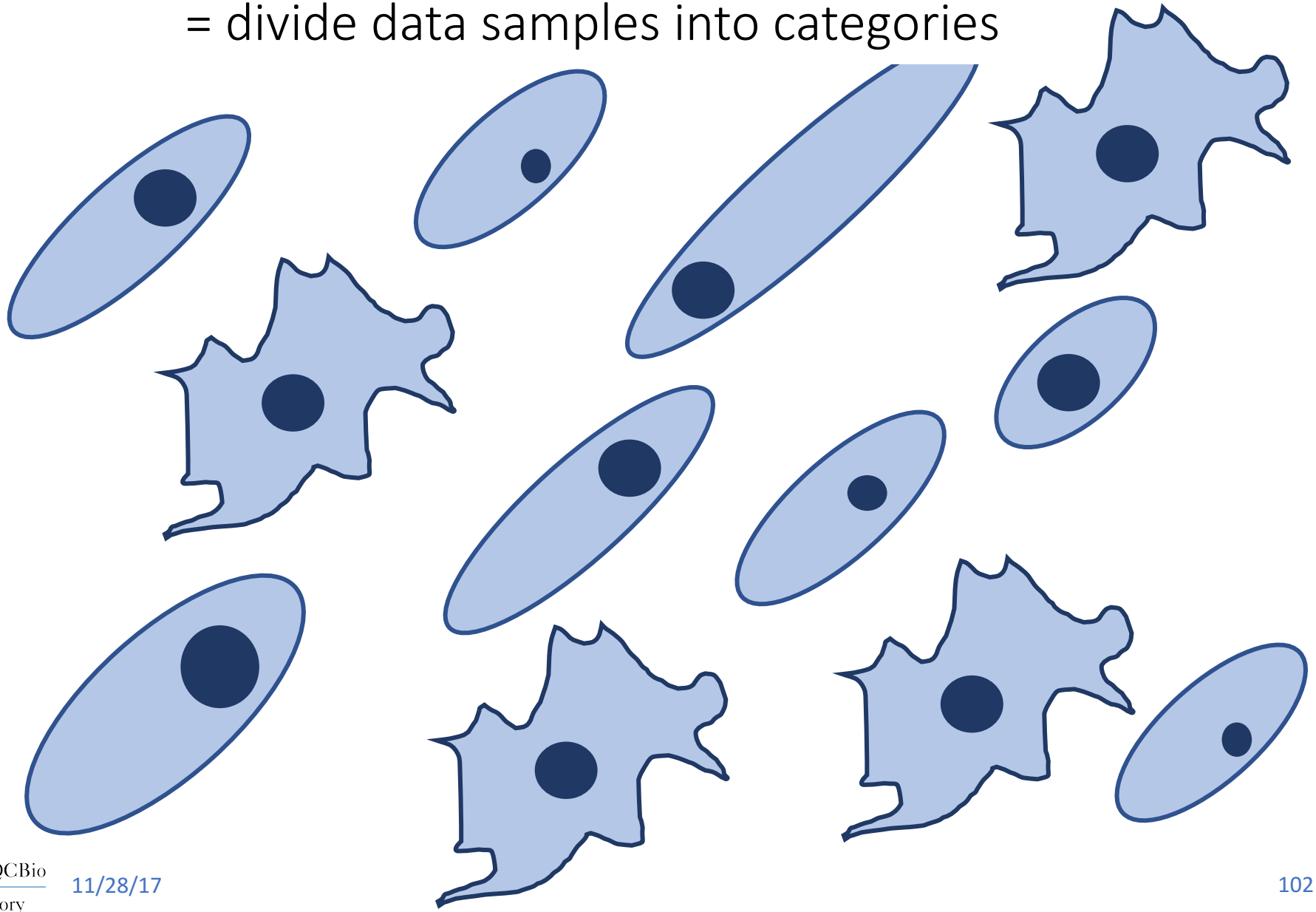


Cruz and Wishart, Cancer Inform. V2 p59 (2006)

ry smoothness

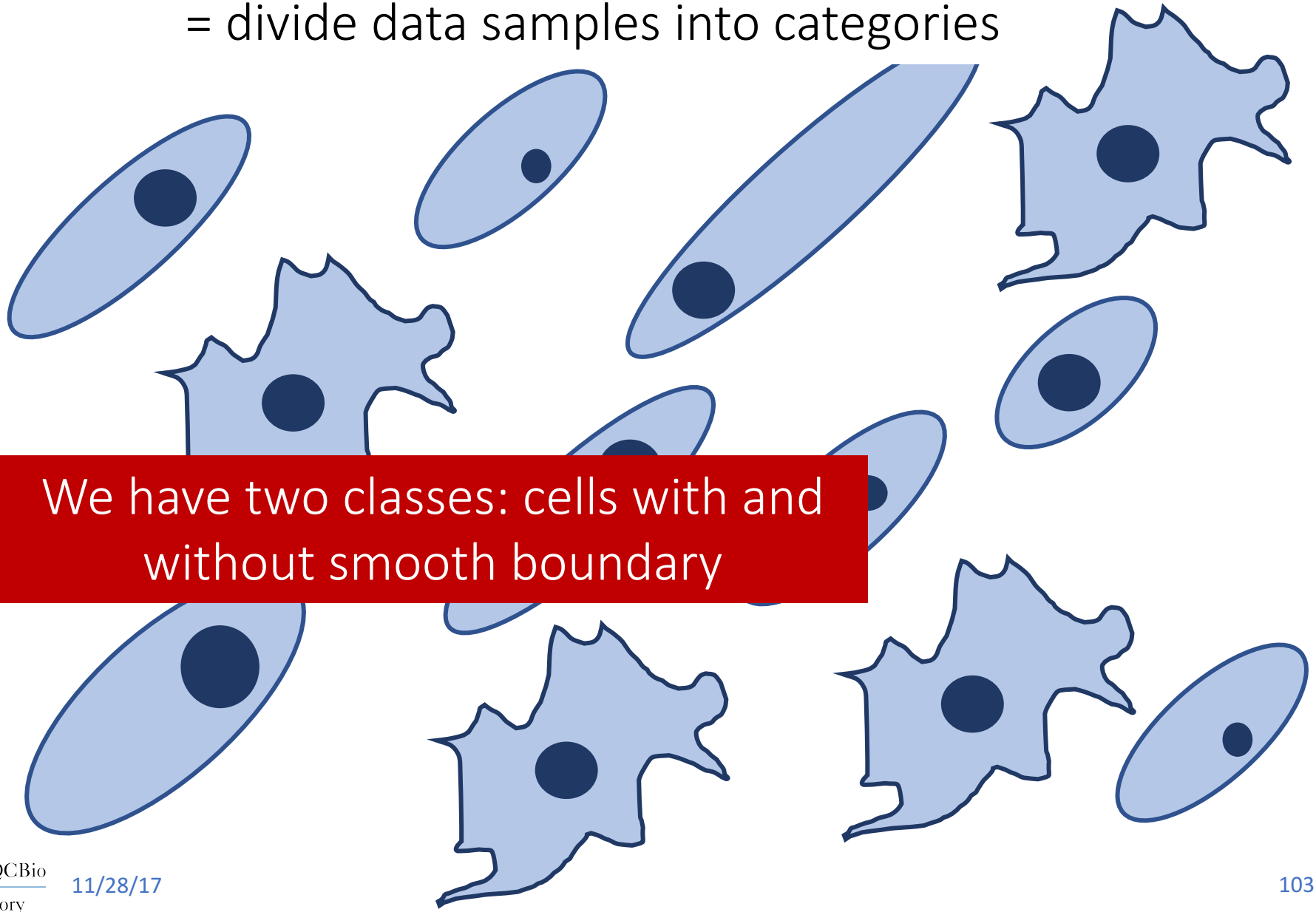
Classification

= divide data samples into categories



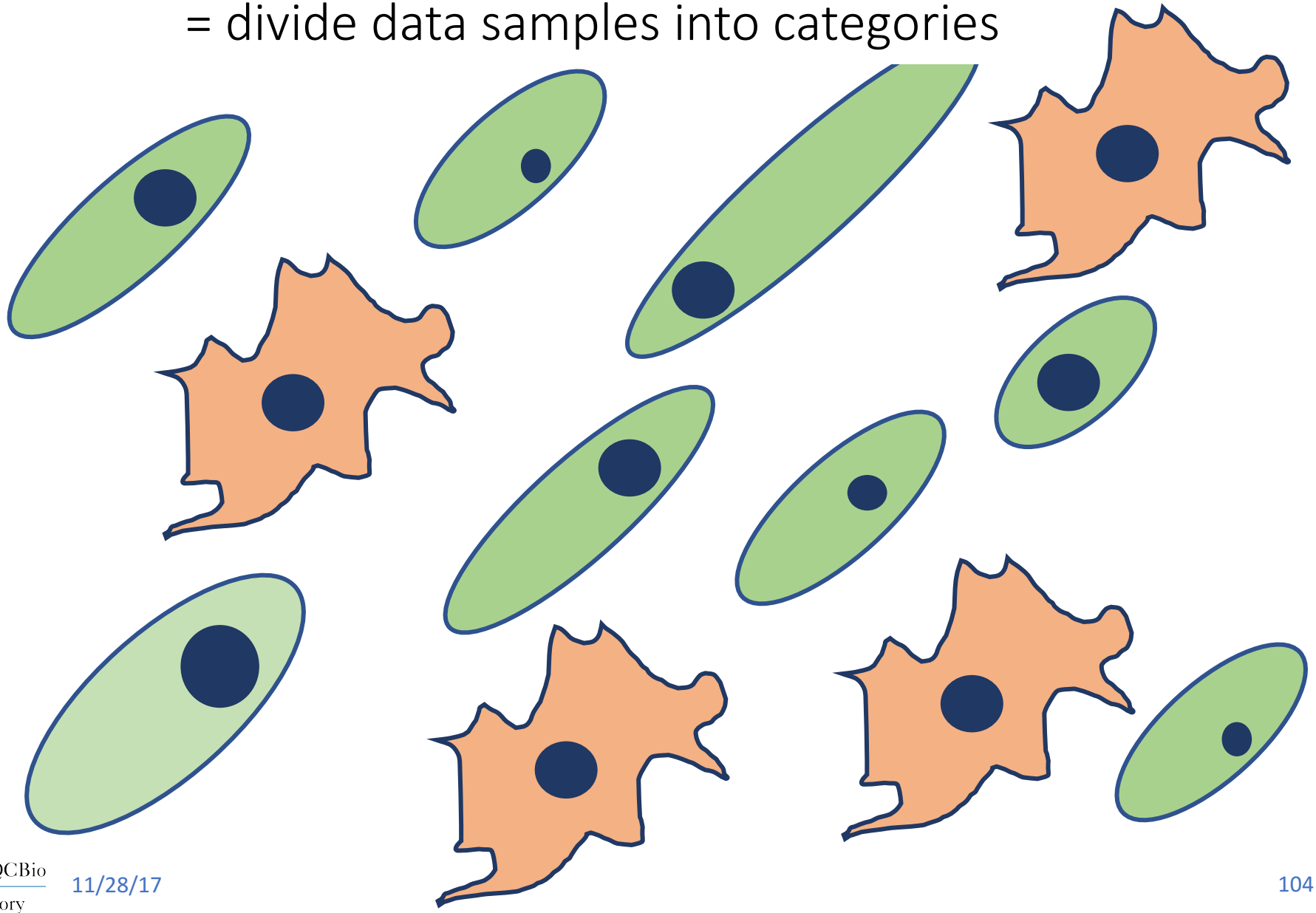
Classification

= divide data samples into categories



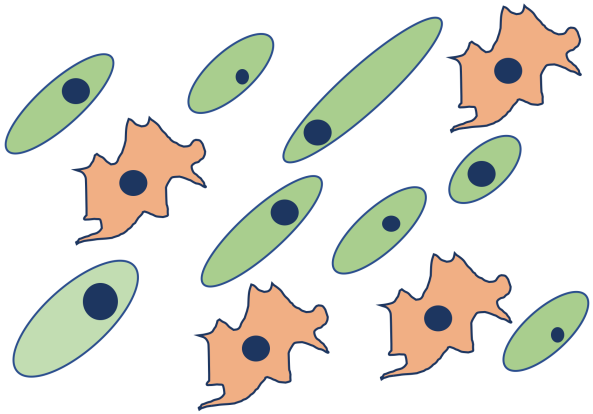
Classification

= divide data samples into categories

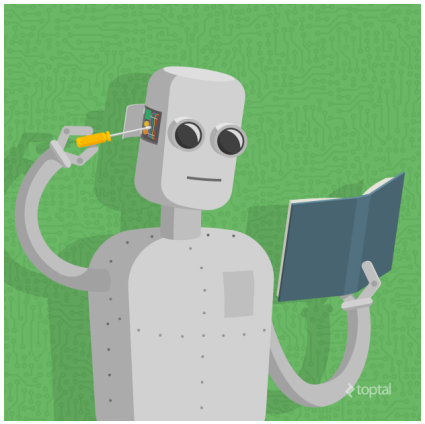


Classification

First the model leans from the examples how to discriminate…



Model learns From examples



Once the model is trained, we can use it to make predictions:

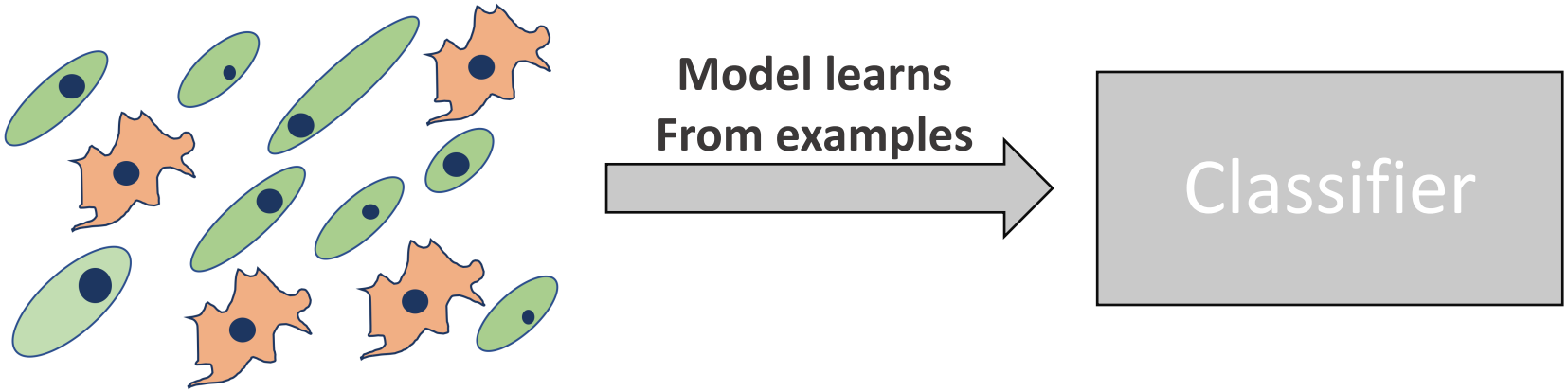
Entry never seen before

Classifier

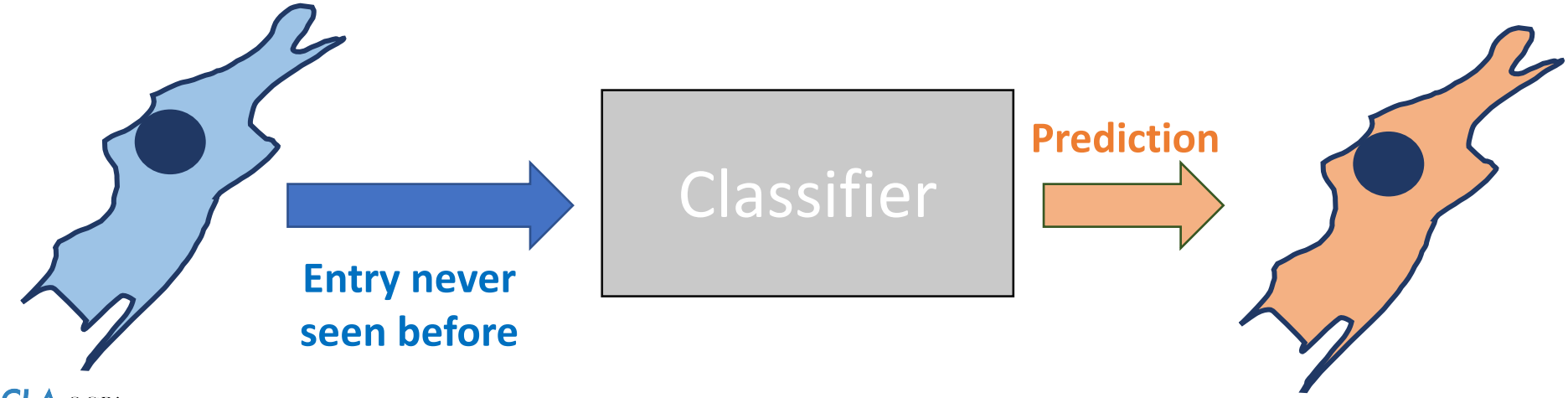
Prediction

Classification

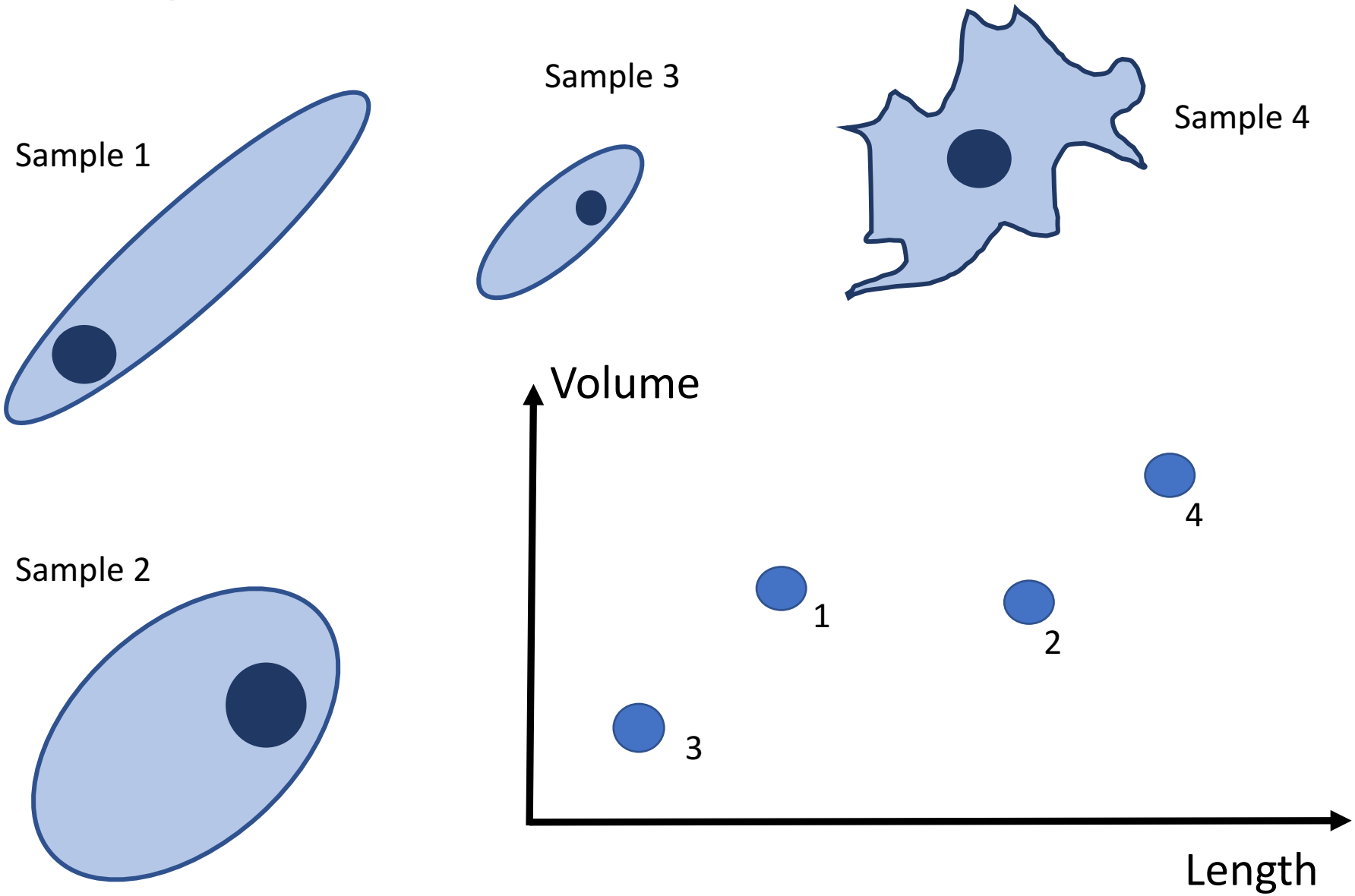
First the model leans from the examples how to discriminate…



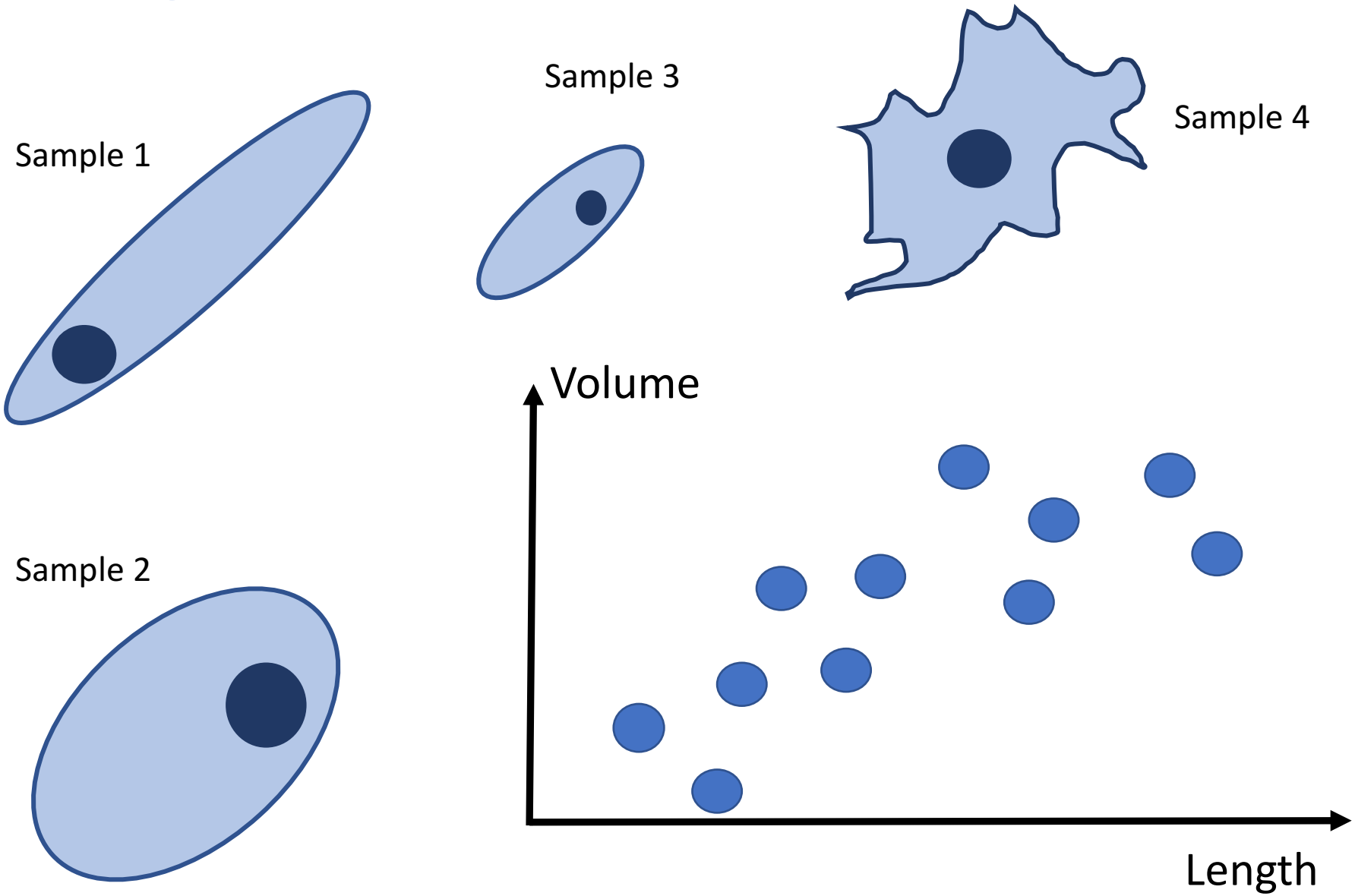
Once the model is trained, we can use it to make predictions:



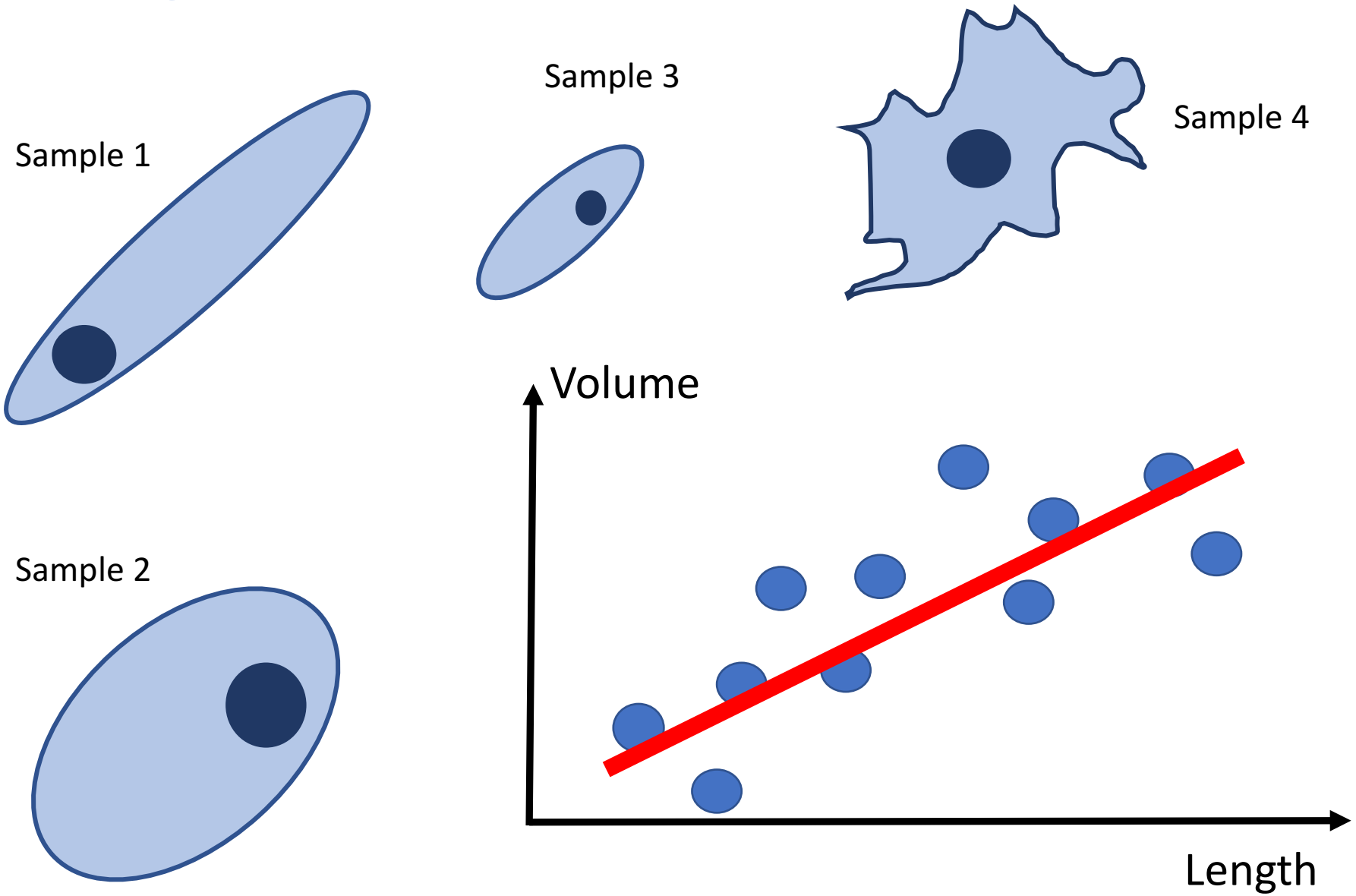
Regression = estimates the value of a continuous variable



Regression = estimates the value of a continuous variable



Regression = estimates the value of a continuous variable



Supervised Learning

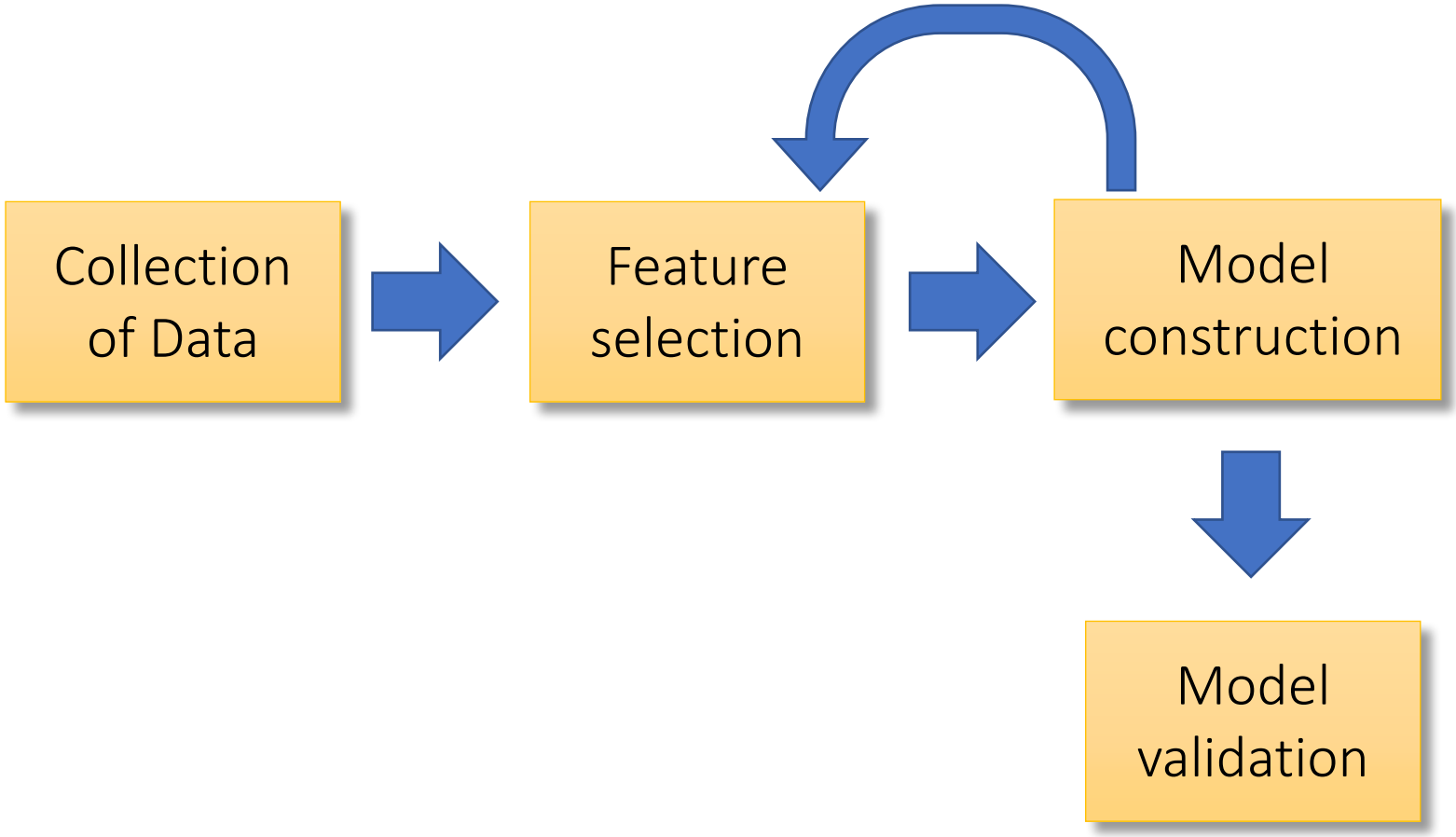
The key characteristic to all supervised learning is that it requires examples from which the model will learn to reproduce the desired patterns

1. In the classification problem, we need a set of cell images and a label that tells to which class they belong to

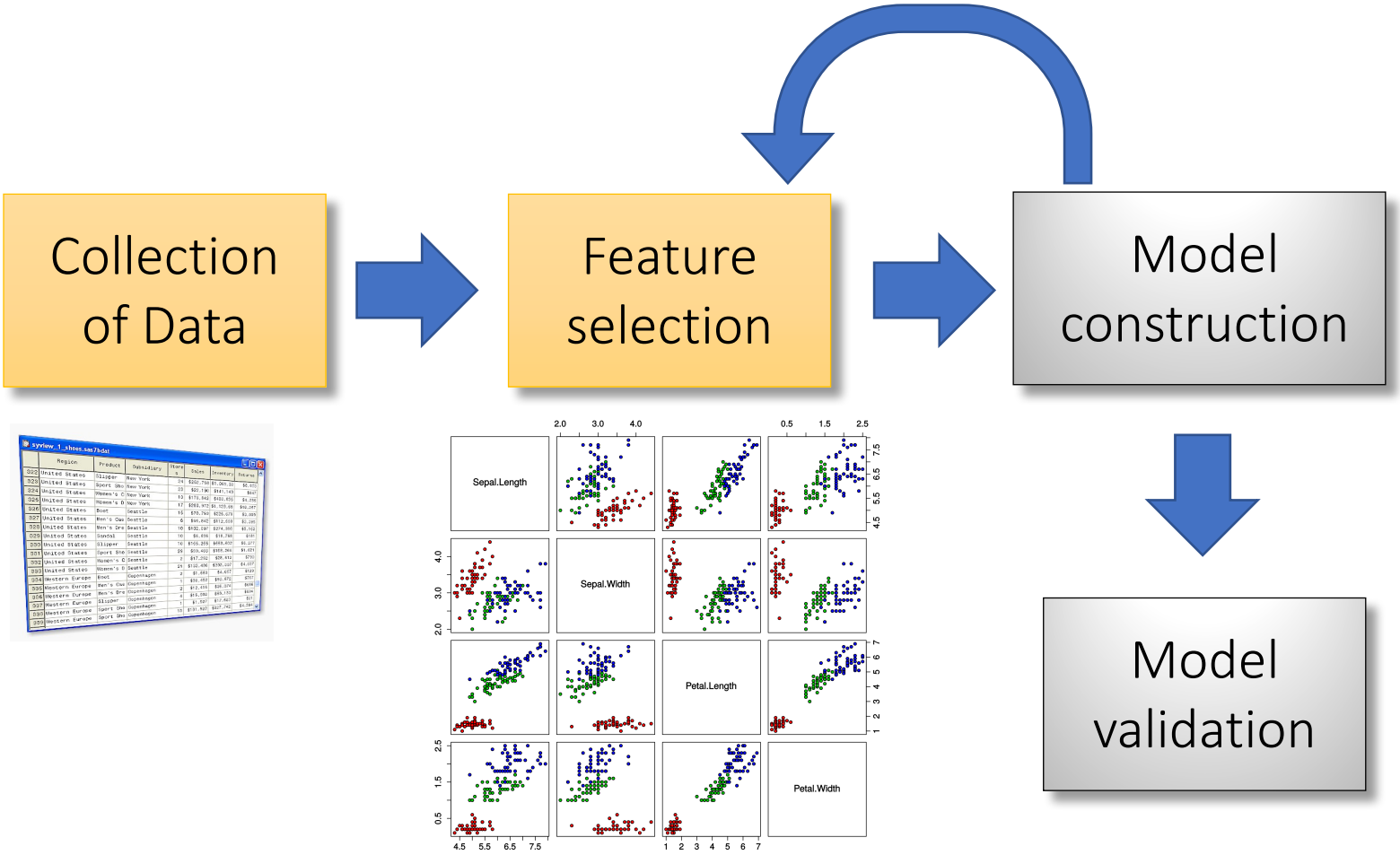
2. In the regression, we need a set of cell images and their associated volume

The data from which the model learns is called training dataset

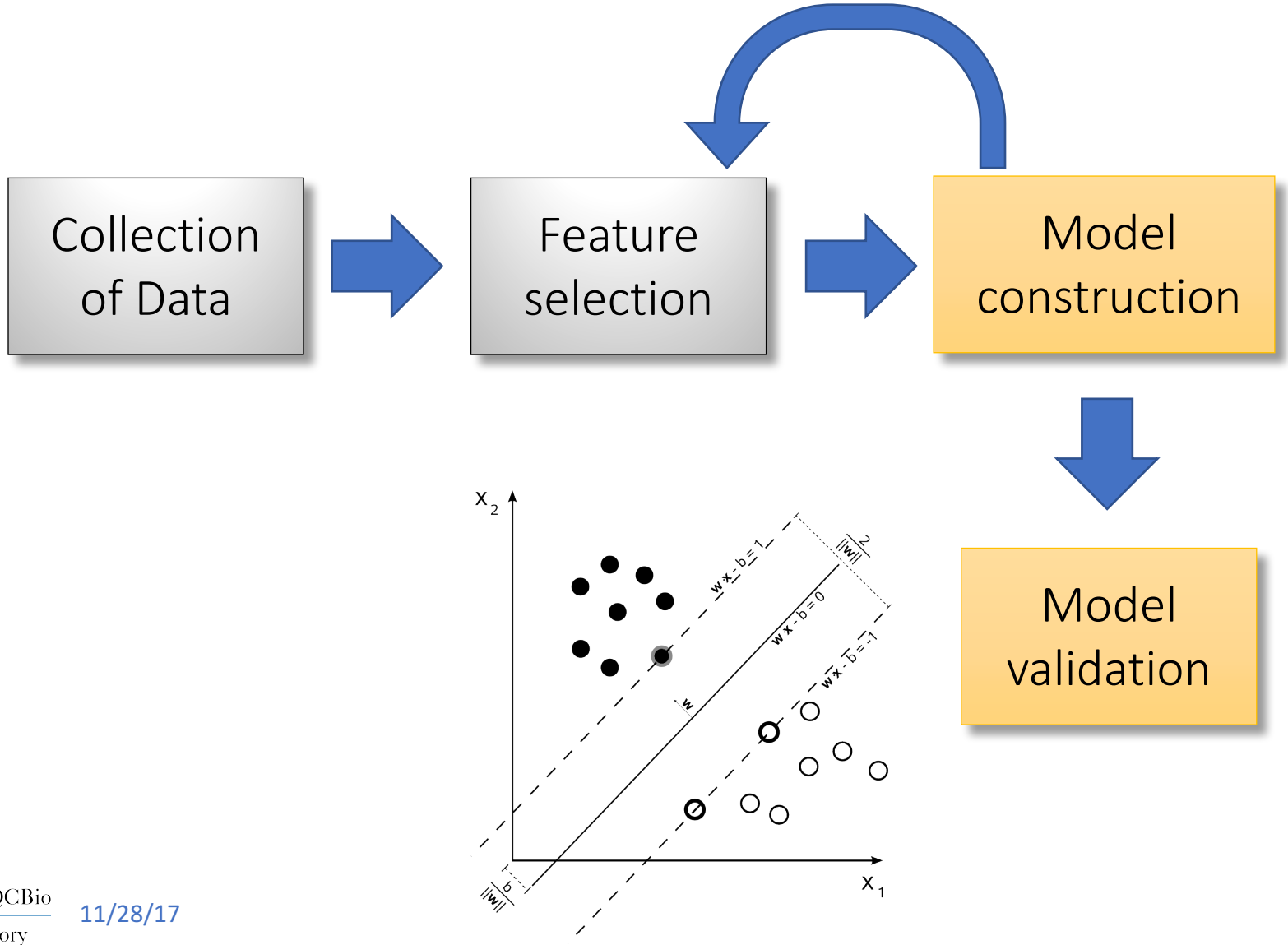
Machine learning pipeline



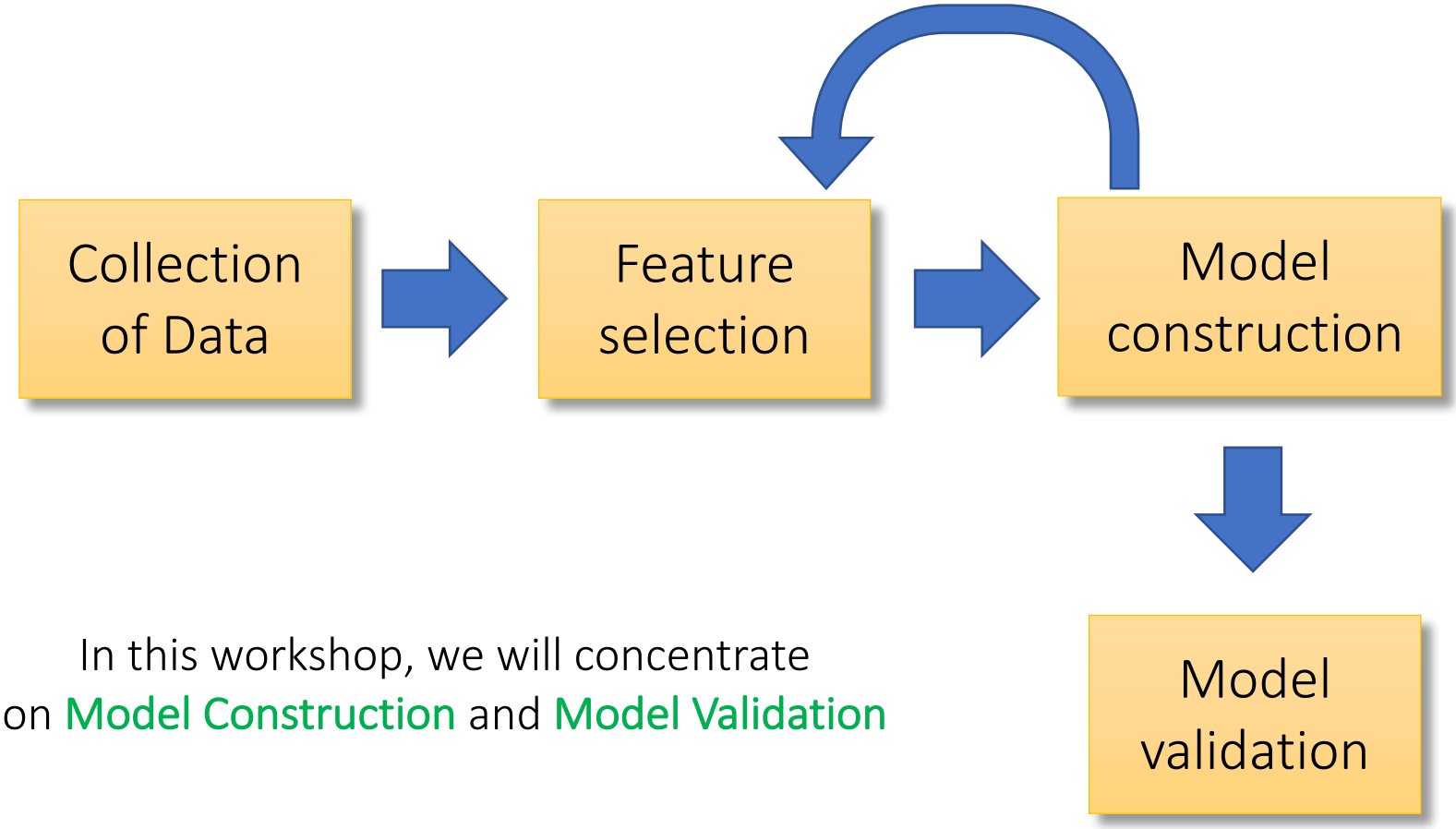
Machine learning pipeline



Machine learning pipeline



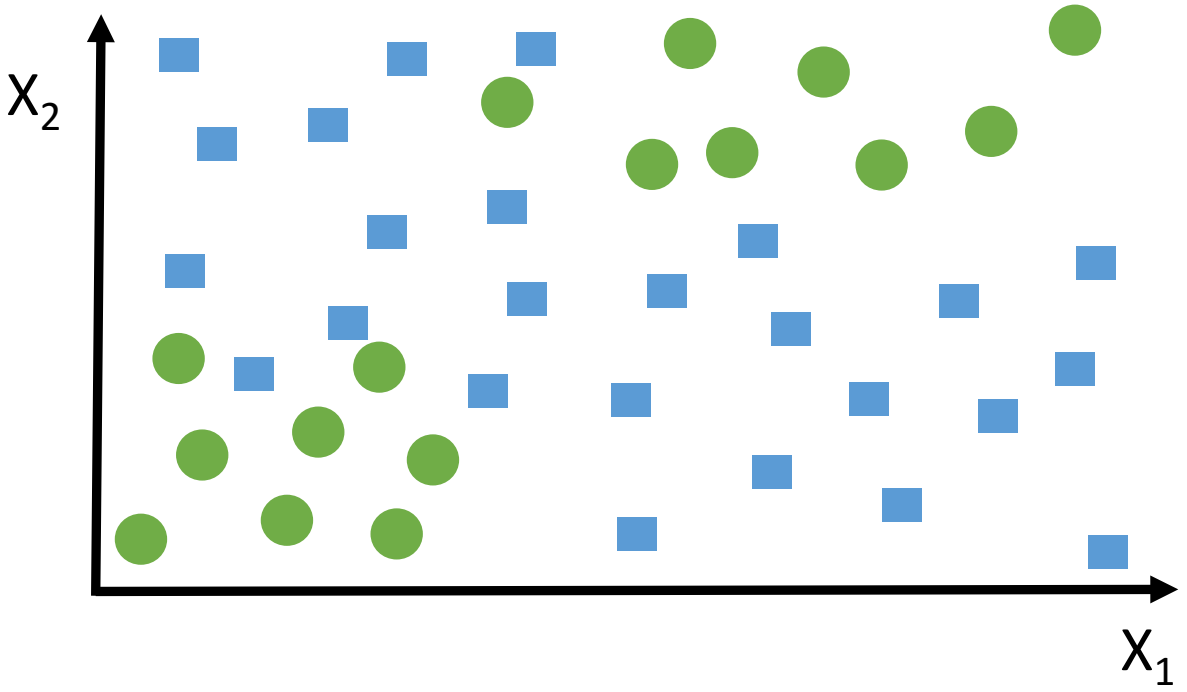
Machine learning pipeline



In this workshop, we will concentrate on Model Construction and Model Validation

Example – Decision trees

Let's introduce how Decision Trees work and use it as a tangible example of what a "model" is



The point is to separate blue squares from green circles

Example – Decision trees

The “learning rule” is simple and has only two steps:

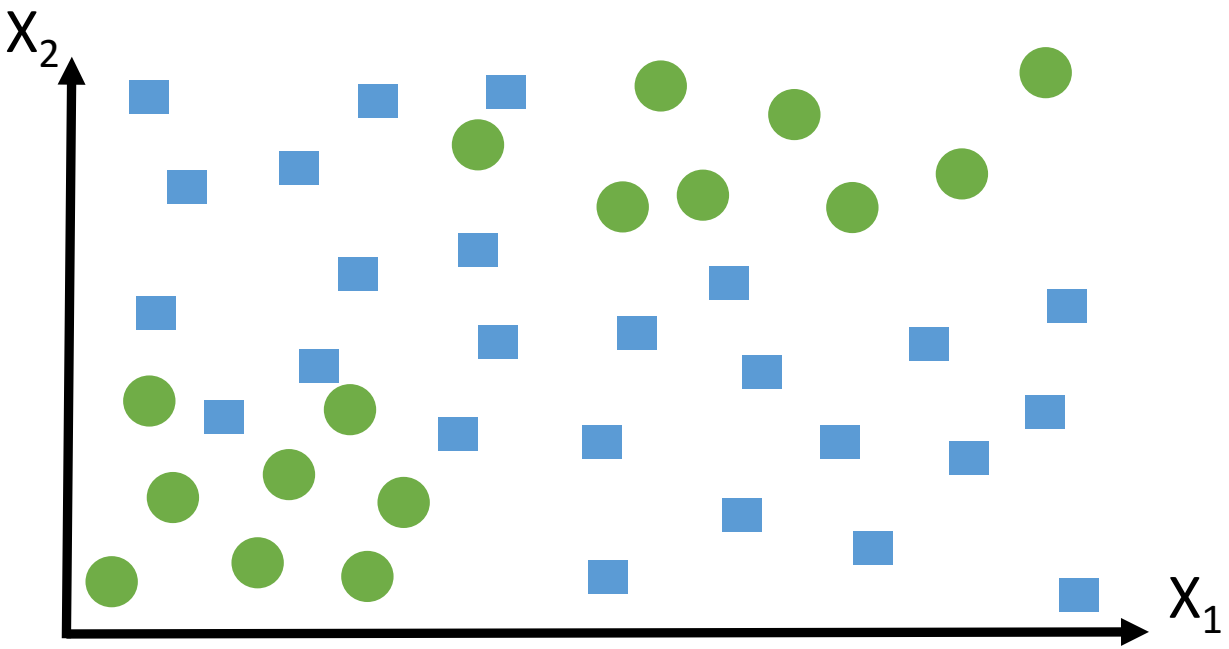
1. Select a feature at random
2. Select a threhsold that separates “the most” the data from each class

Steps 1 and 2 are repeated until a satisfactory model is achieved (aka stopping conditions).

Although this is a simplification of how Decision trees, it is a useful analogy and we can follow it without spending a lot of time

Example

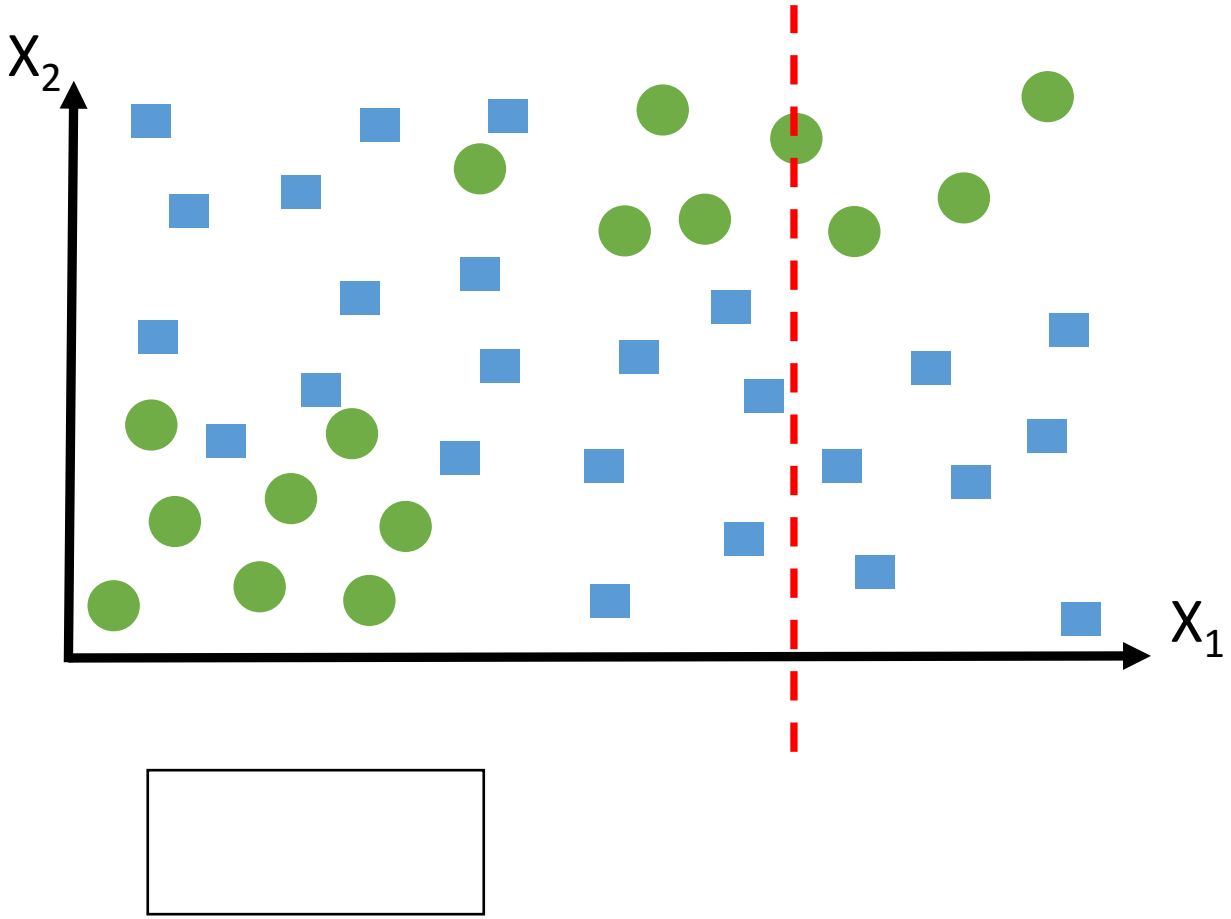
Let's start with X_1



Example

Let's start with X_1

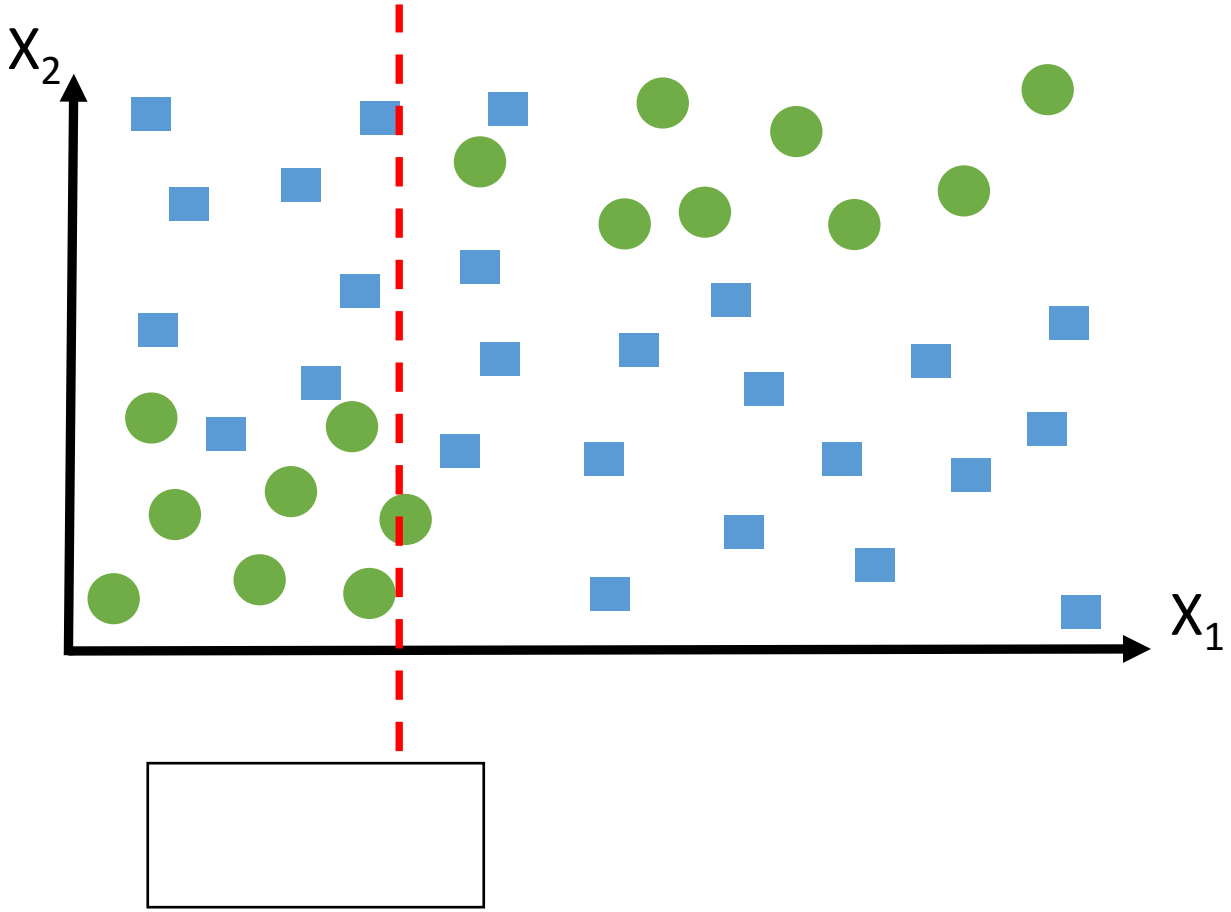
Select a threshold



Example

Let's start with X_1

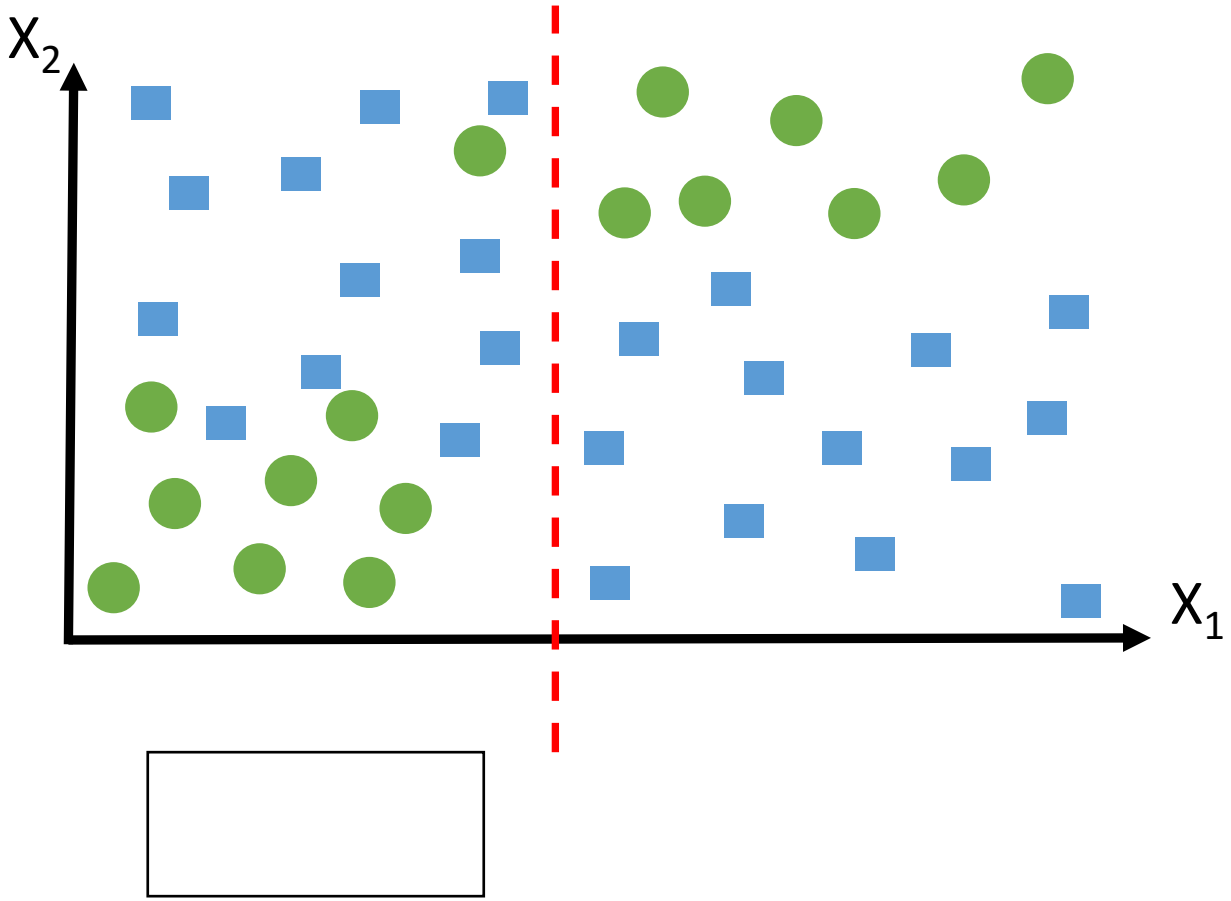
Select a threshold



Example

Let's start with X_1

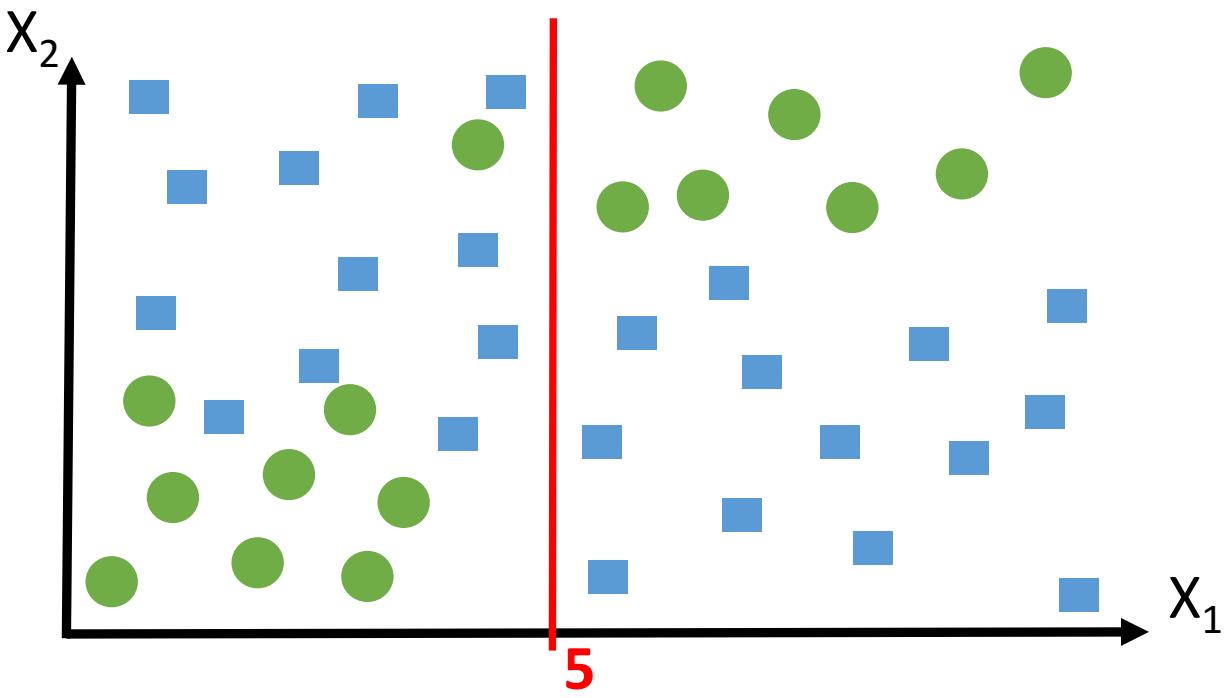
Select a threshold

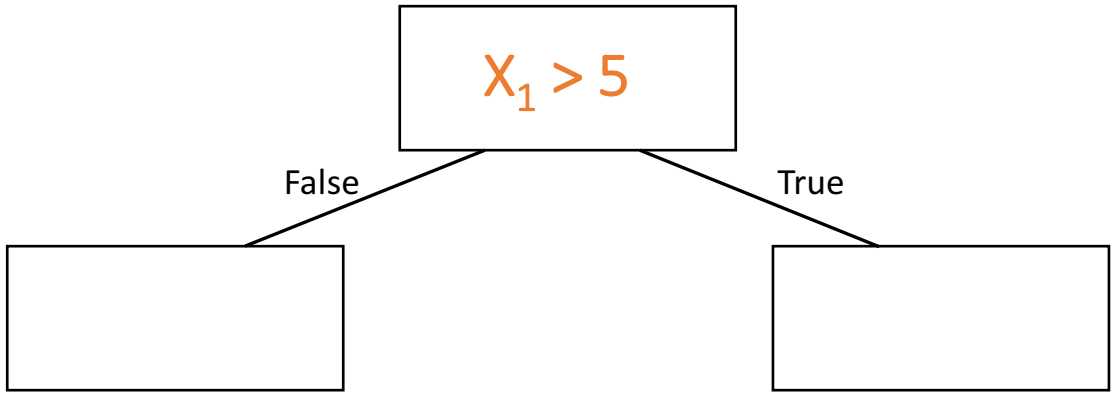


Example

Let's start with X_1

Select a threshold

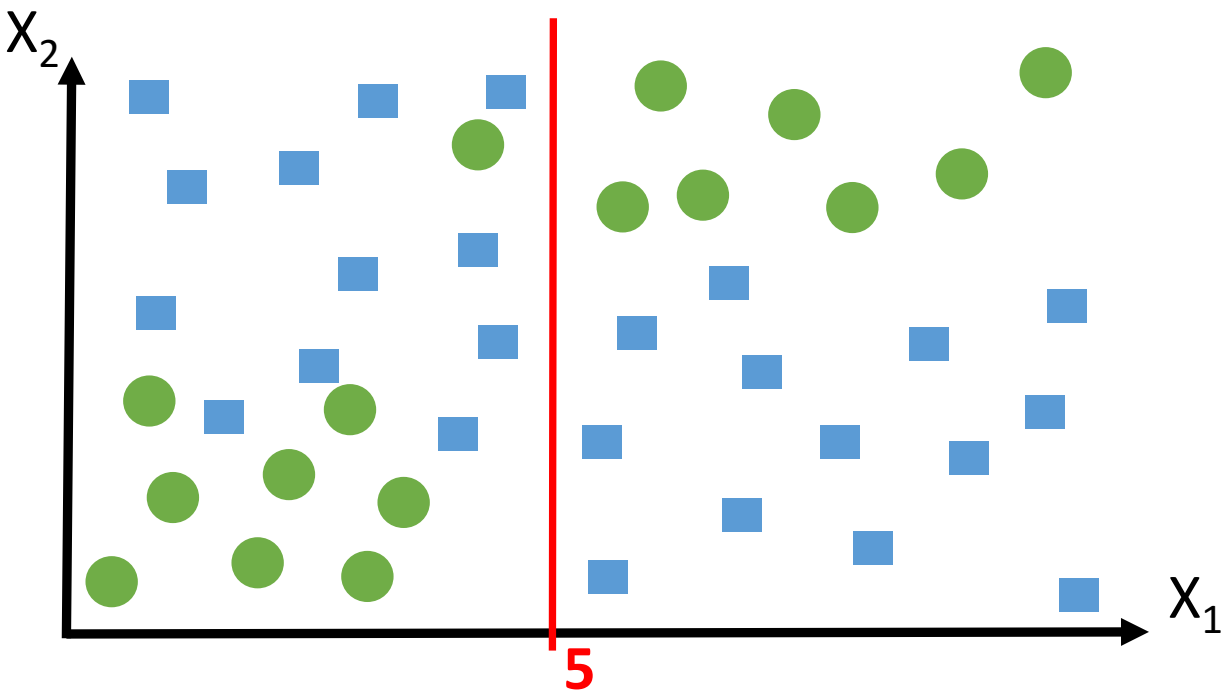


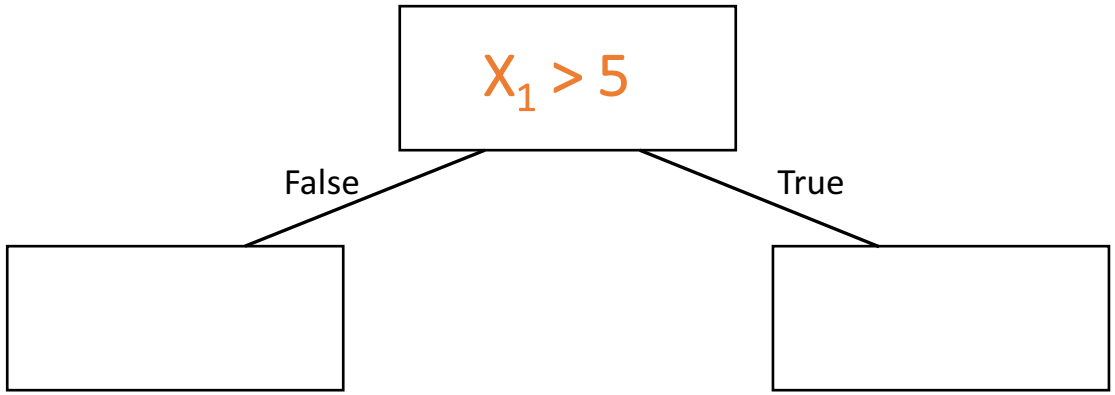


Example

For each branch…

Let's start with X_2



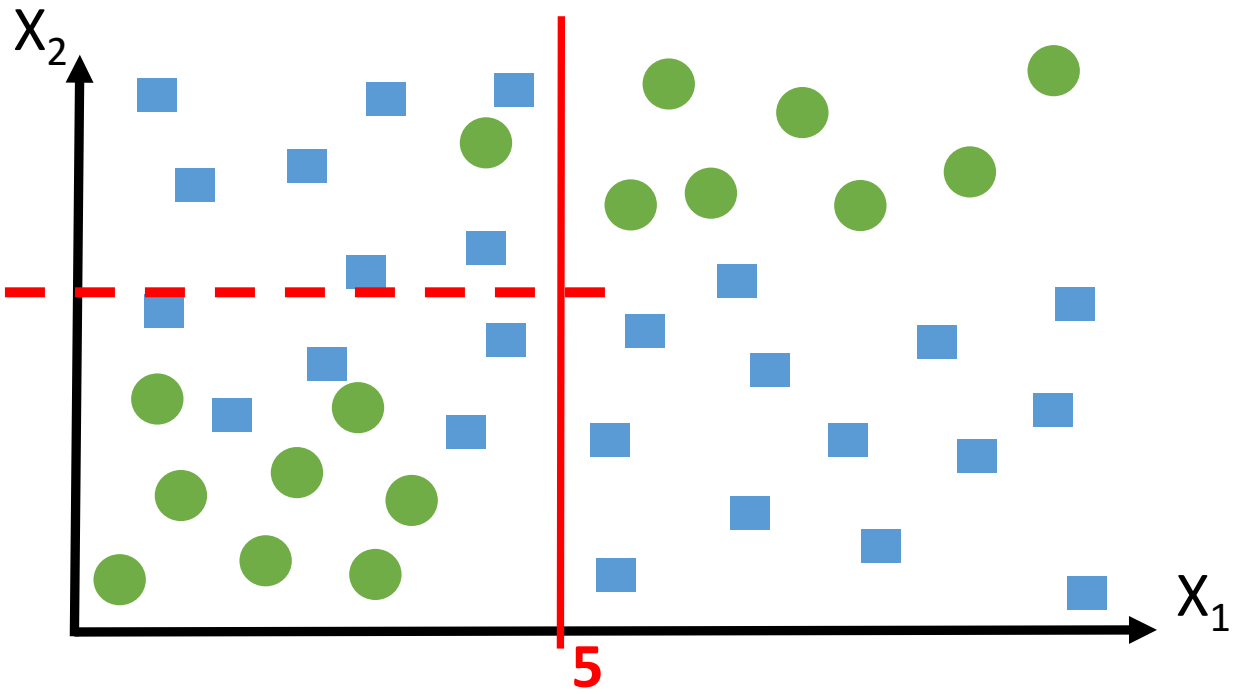


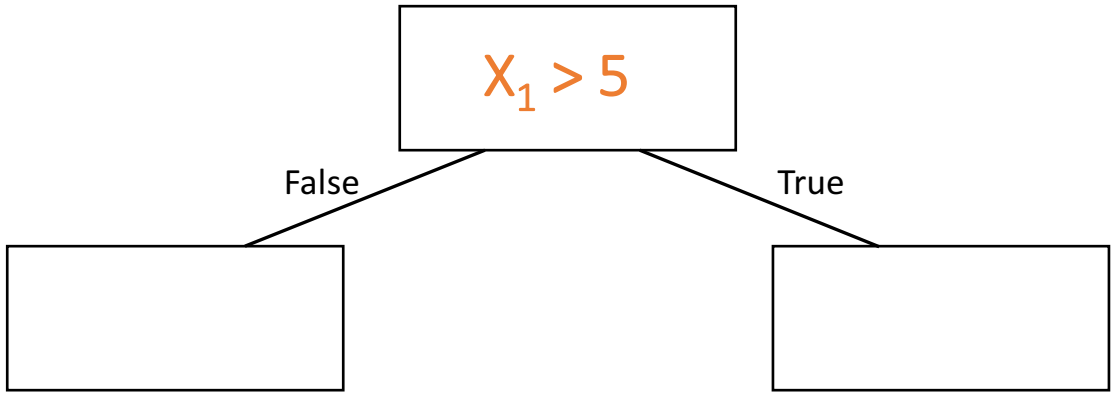
Example

For each branch…

Let's start with X_2

Select a threshold



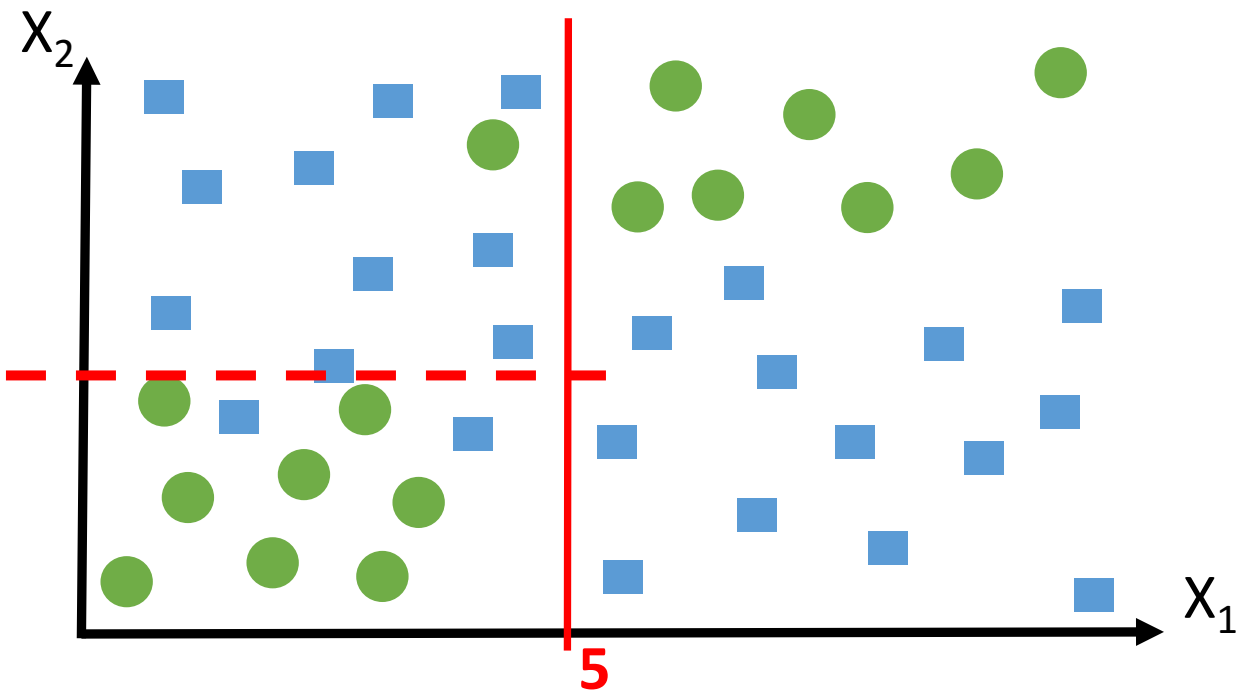


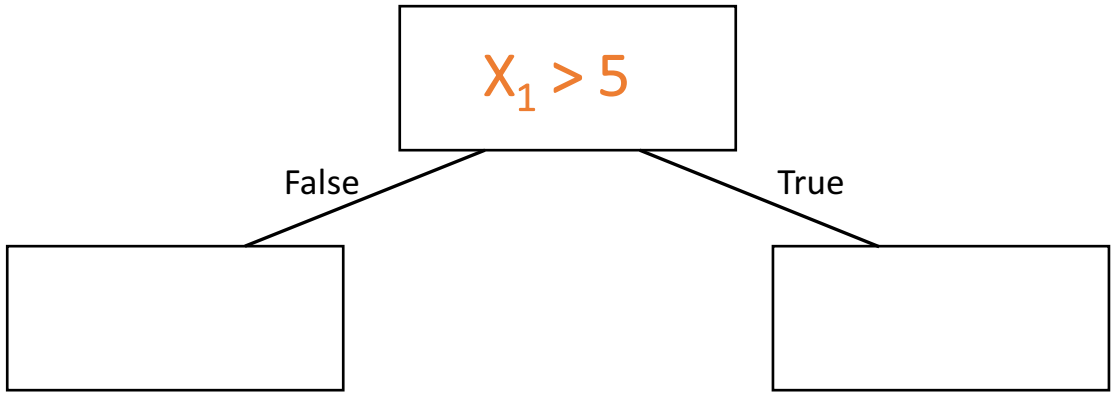
Example

For each branch…

Let's start with X_2

Select a threshold



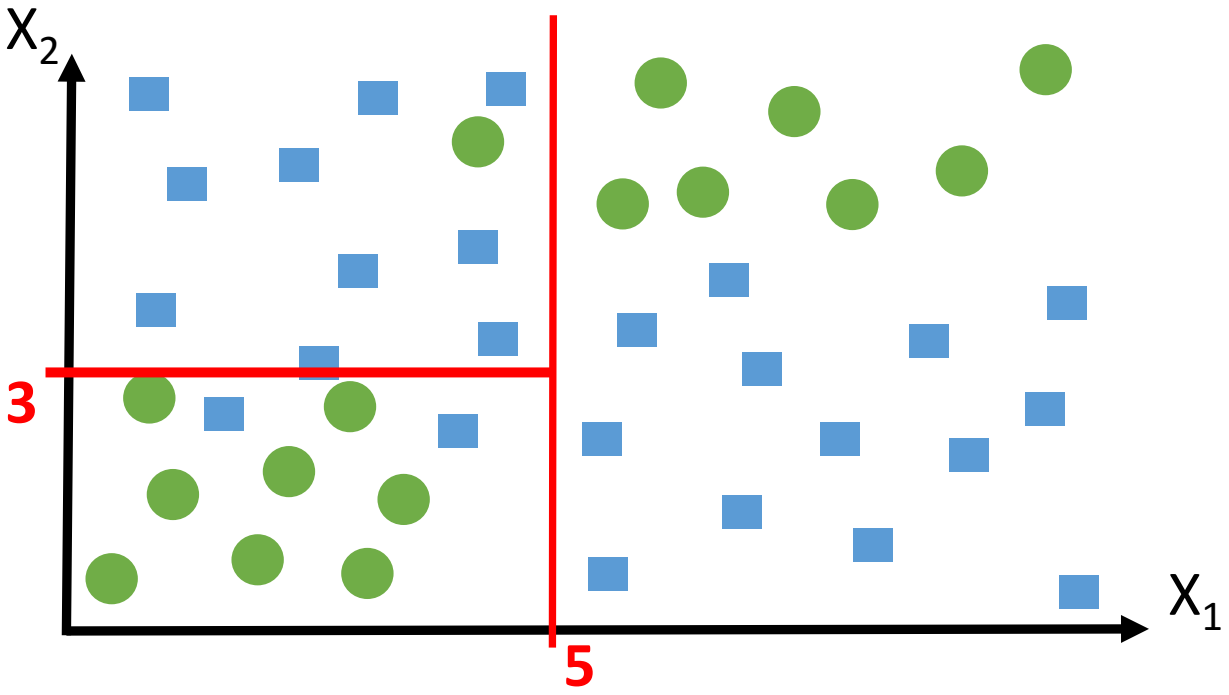


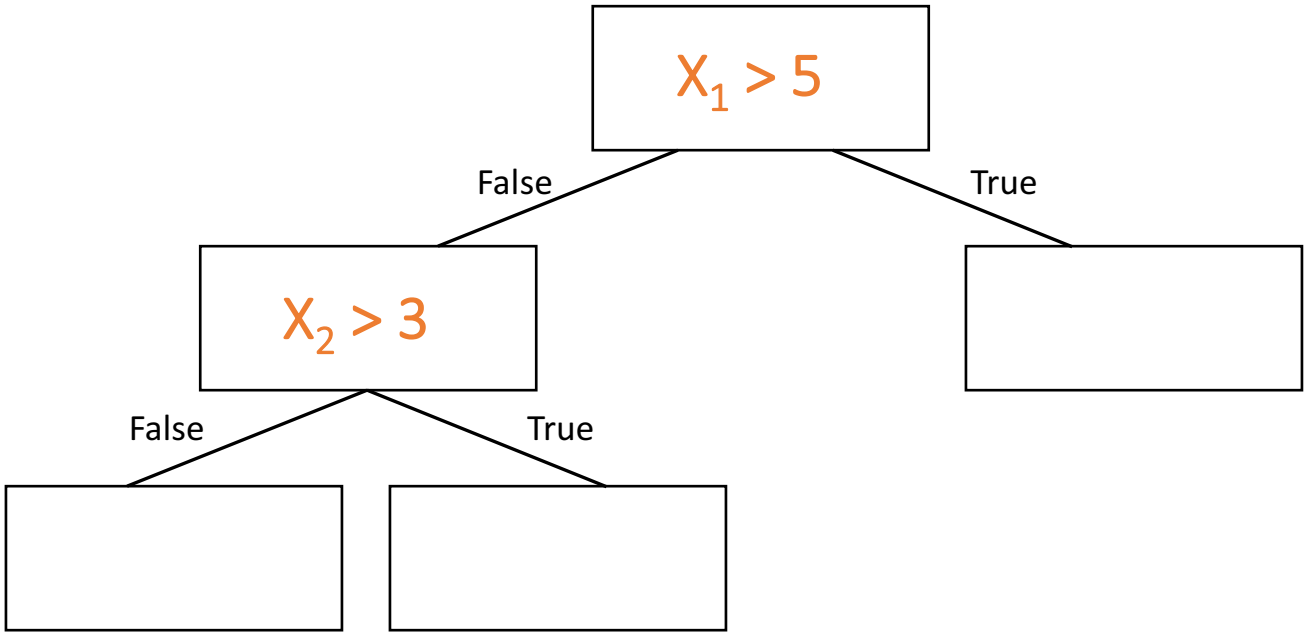
Example

For each branch…

Let's start with X_2

Select a threshold



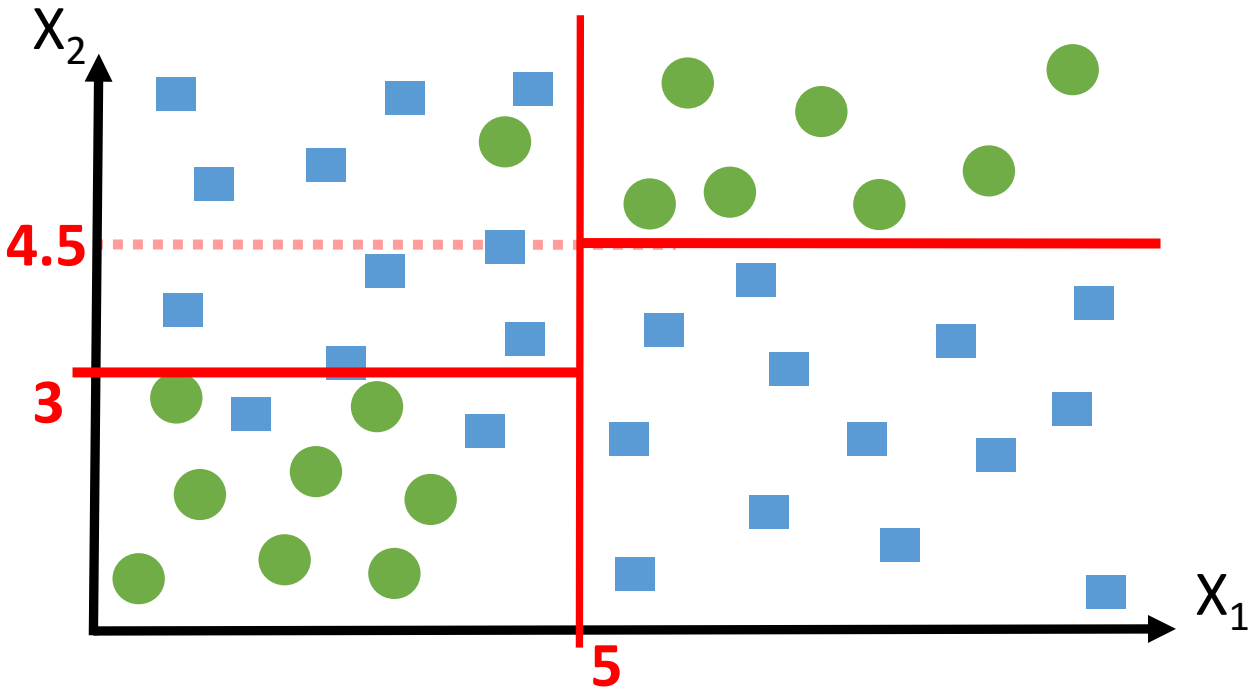


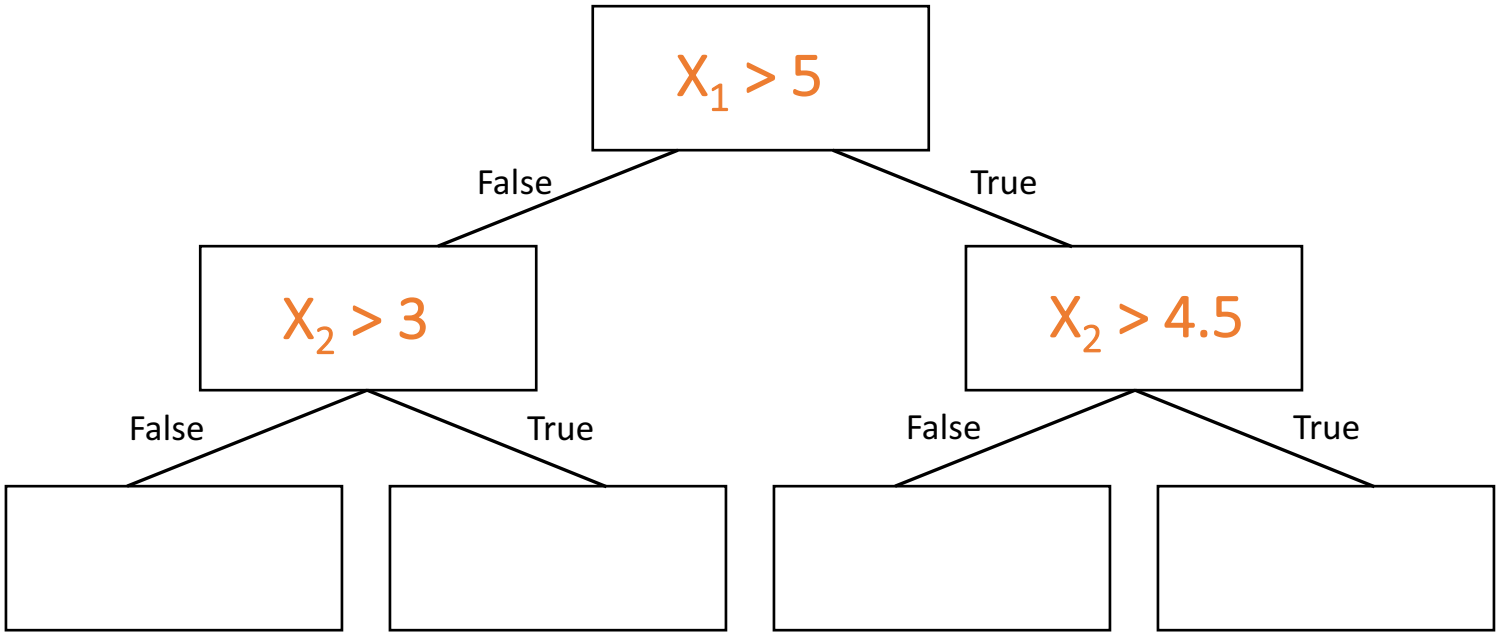
Example

For each branch…

Let's start with X_2

Select a threshold



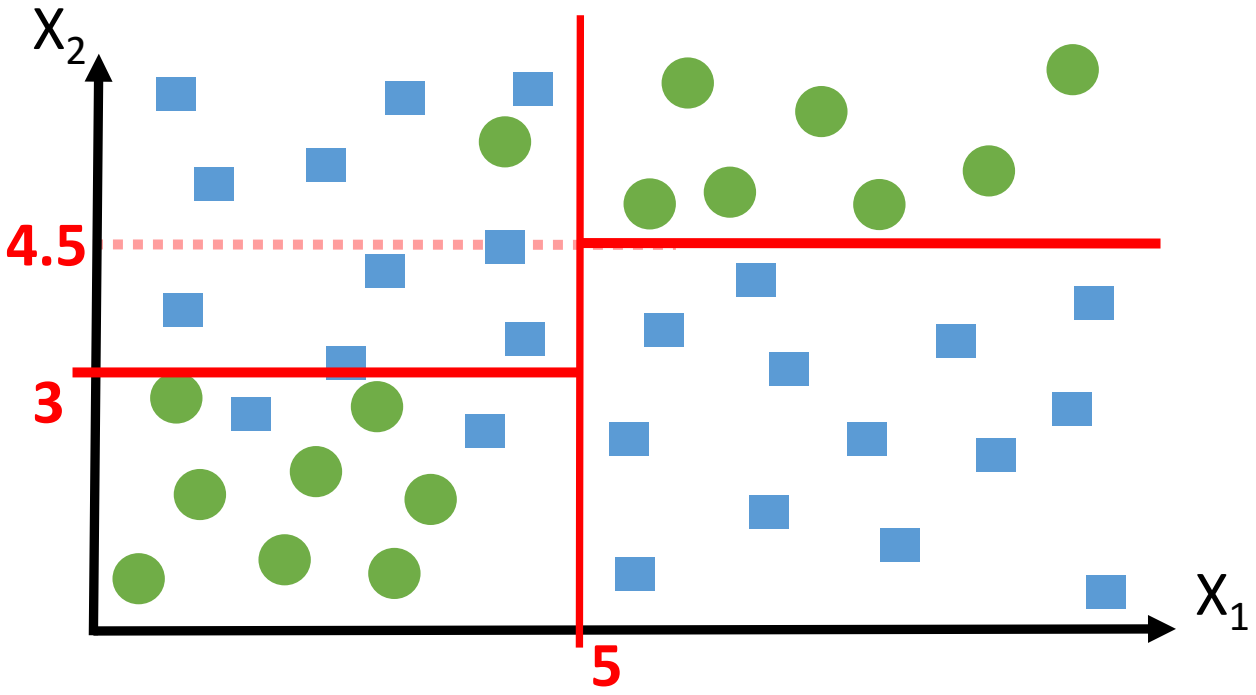


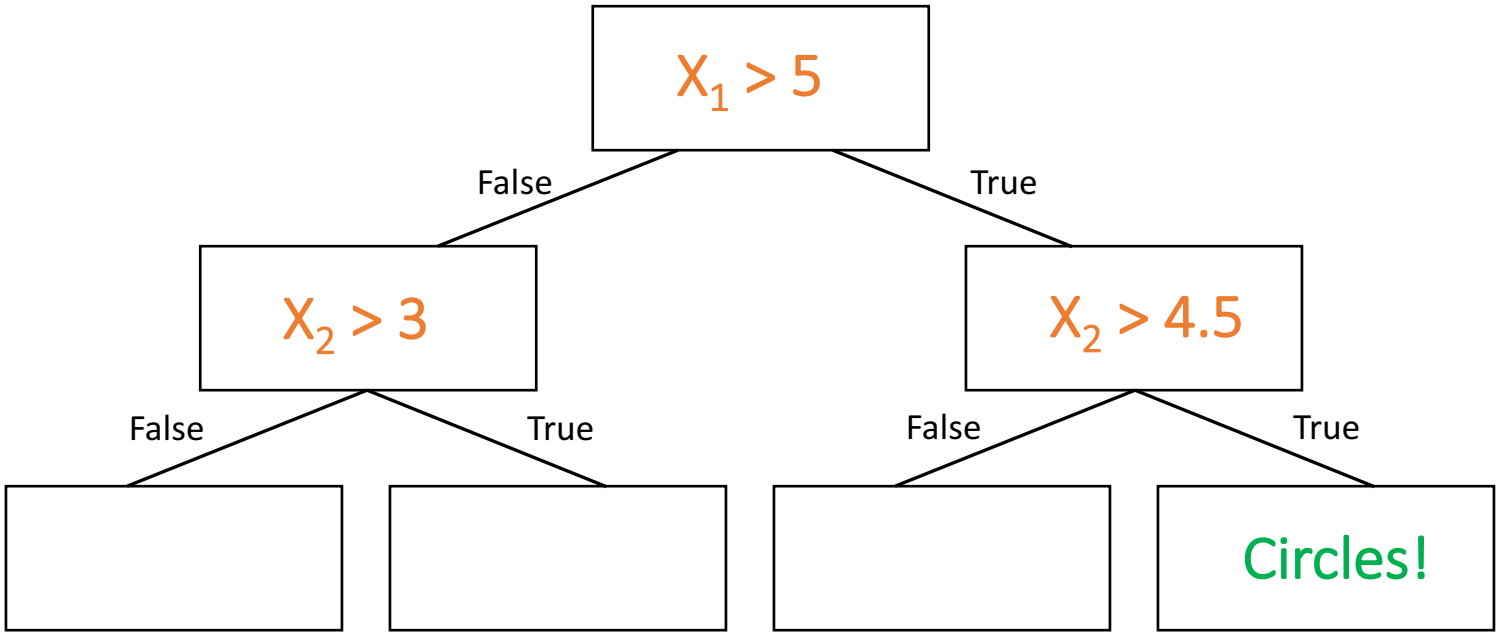
Example

For each branch…

Let's start with X_2

Select a threshold



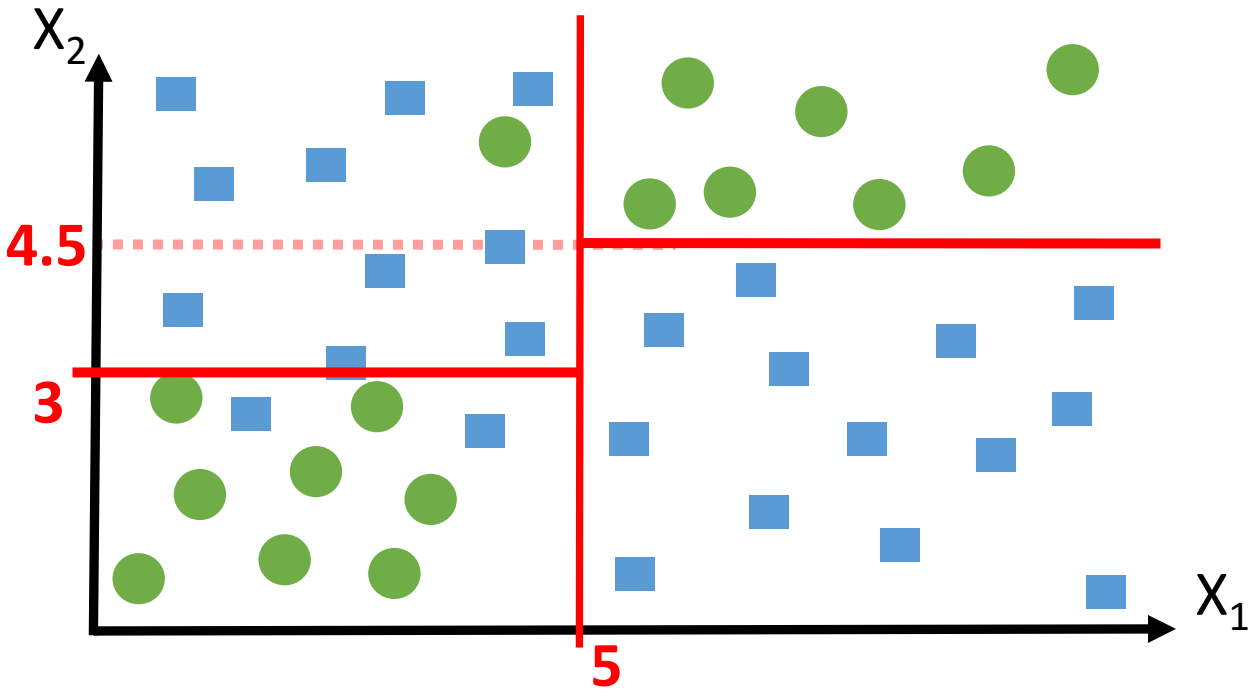


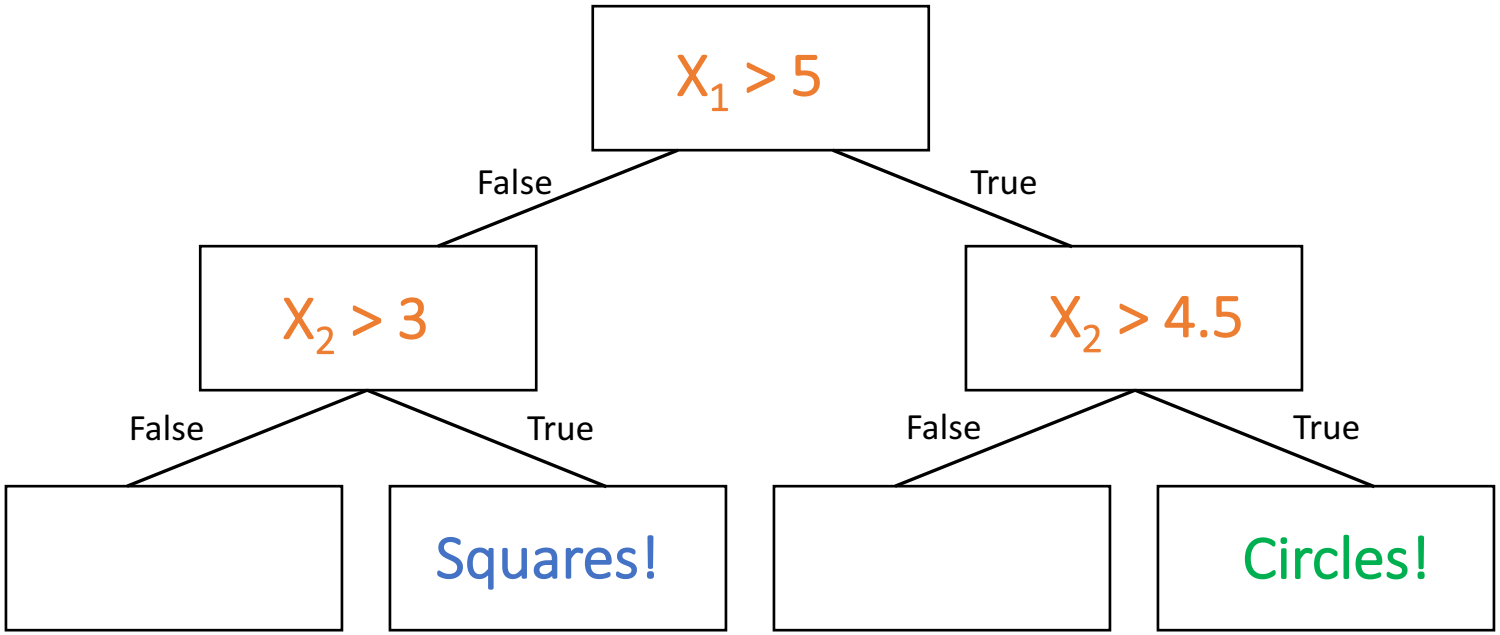
Example

For each branch…

Let's start with X_2

Select a threshold



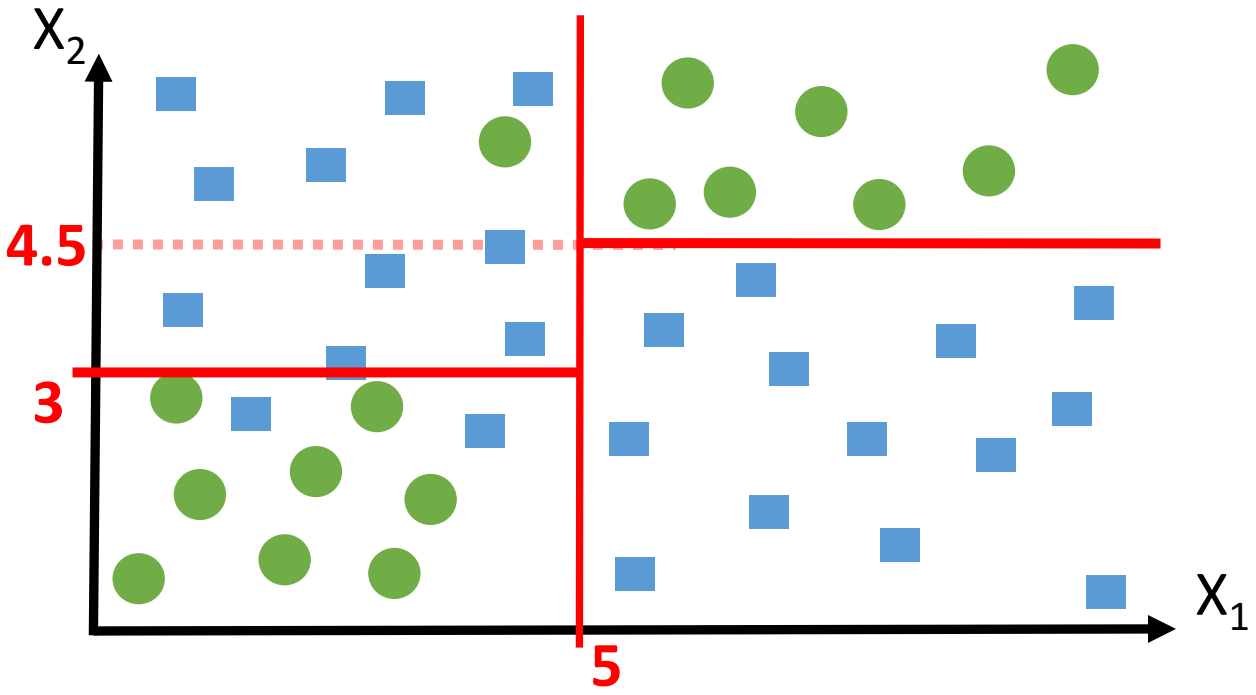


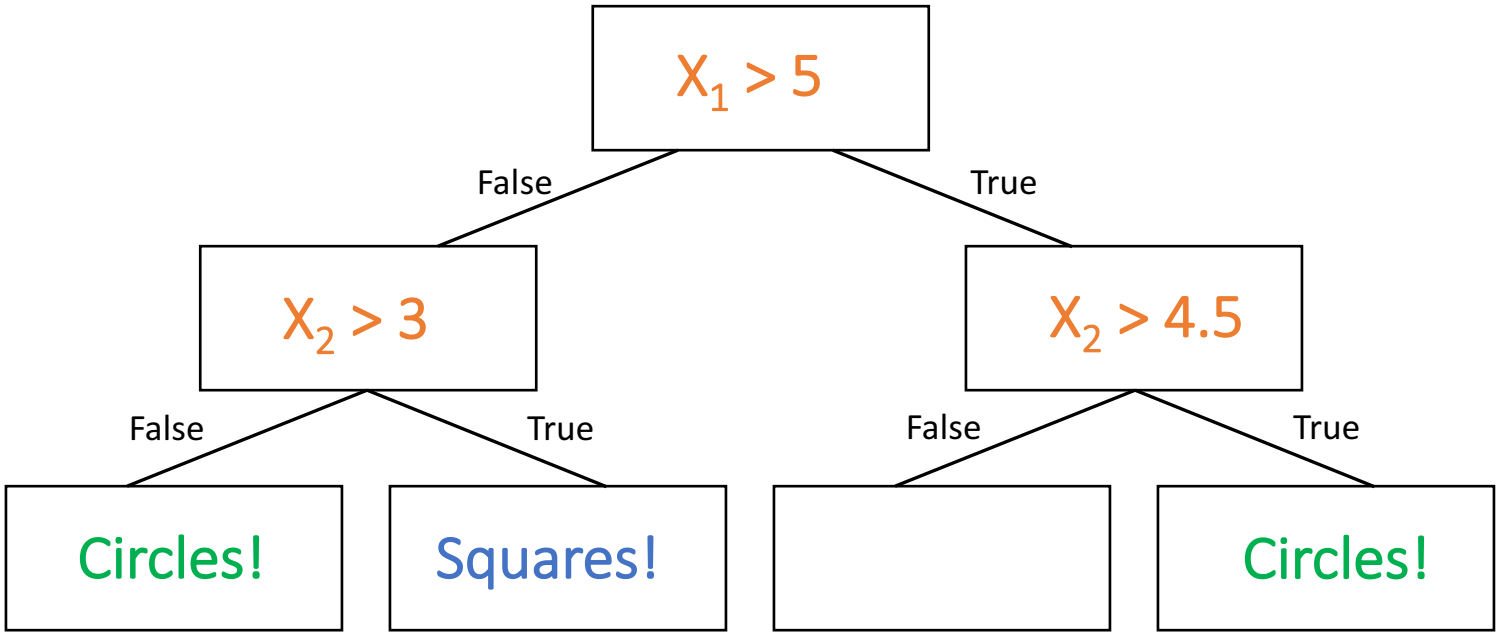
Example

For each branch…

Let's start with X_2

Select a threshold



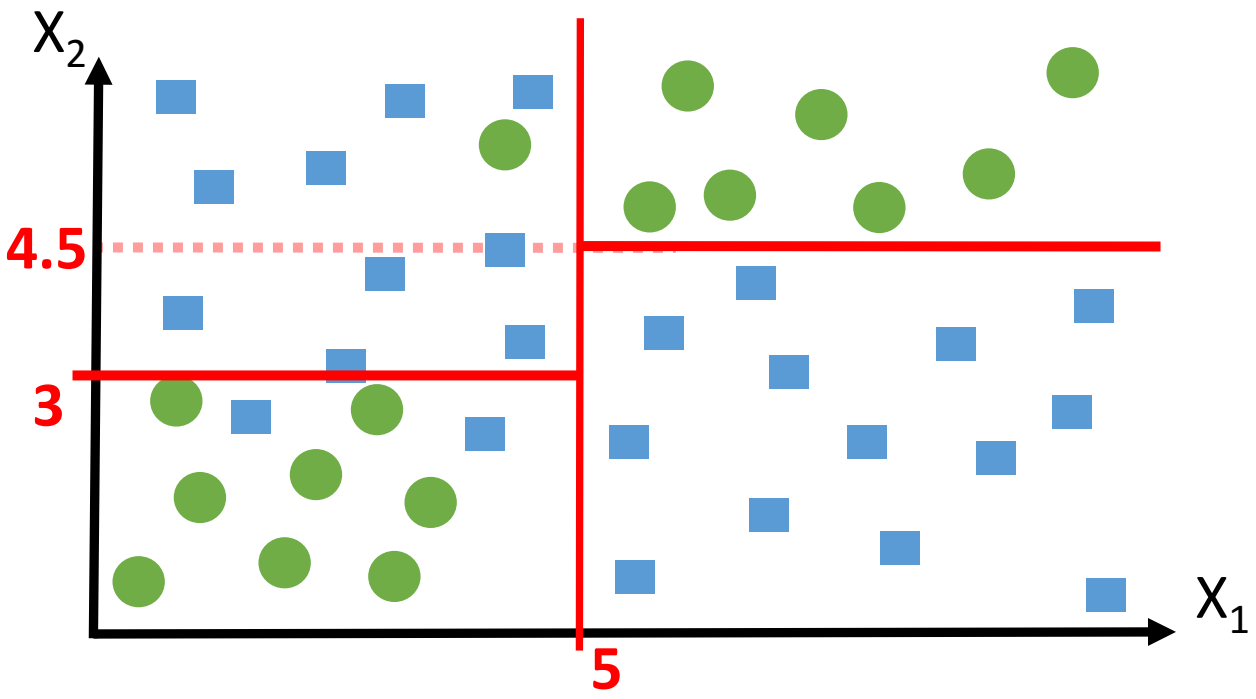


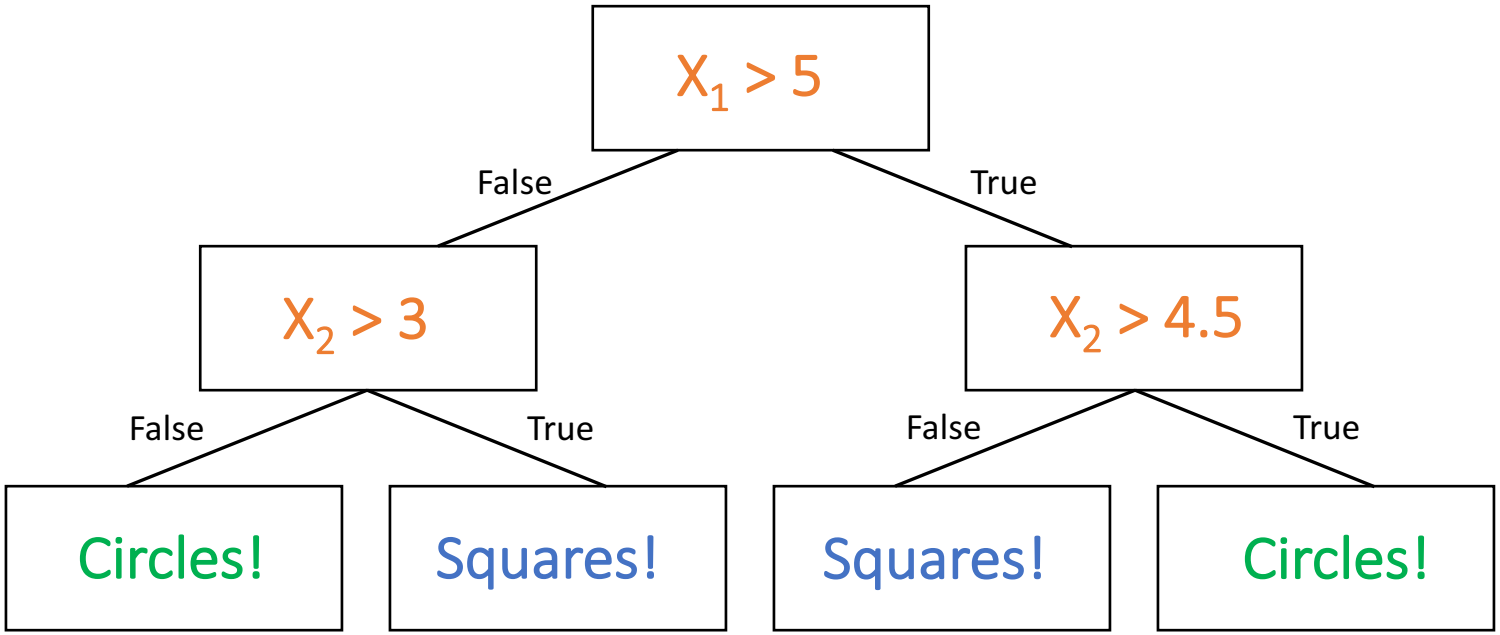
Example

For each branch…

Let's start with X_2

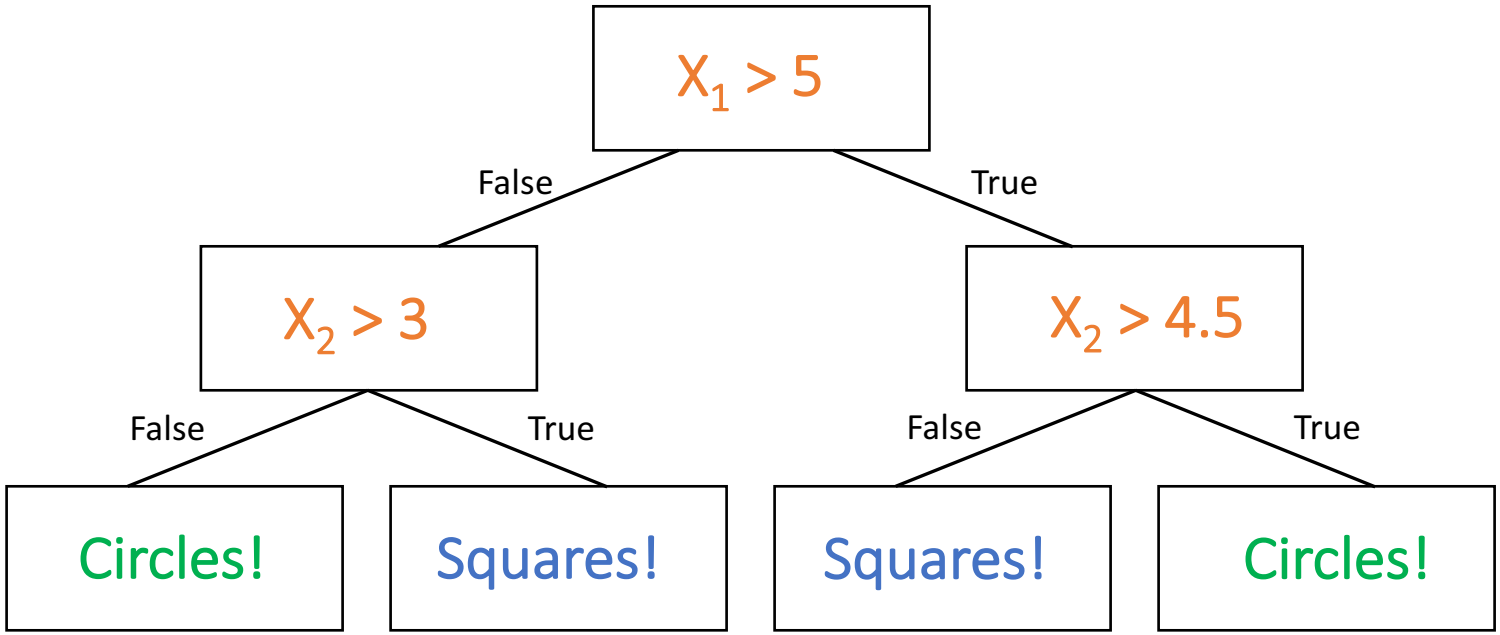
Select a threshold





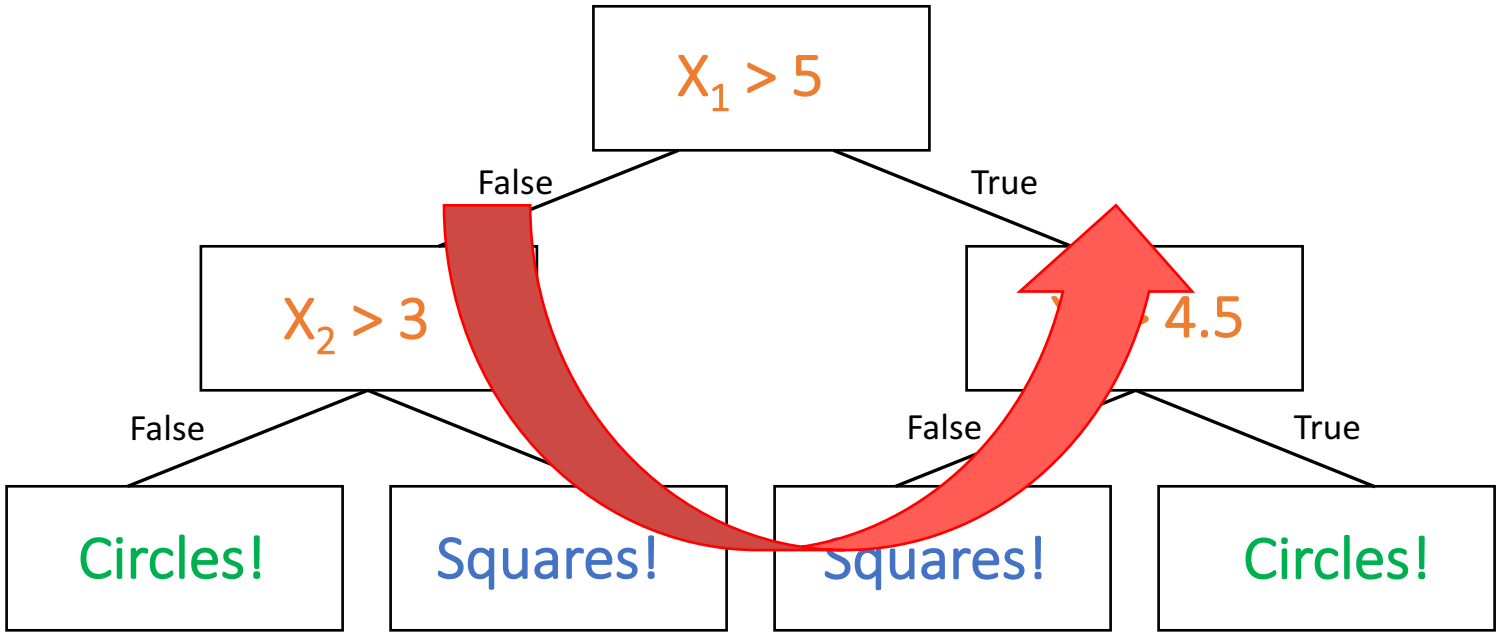
Decision tree

We just created our first our first classifier!



Decision tree

We just created our first our first classifier!

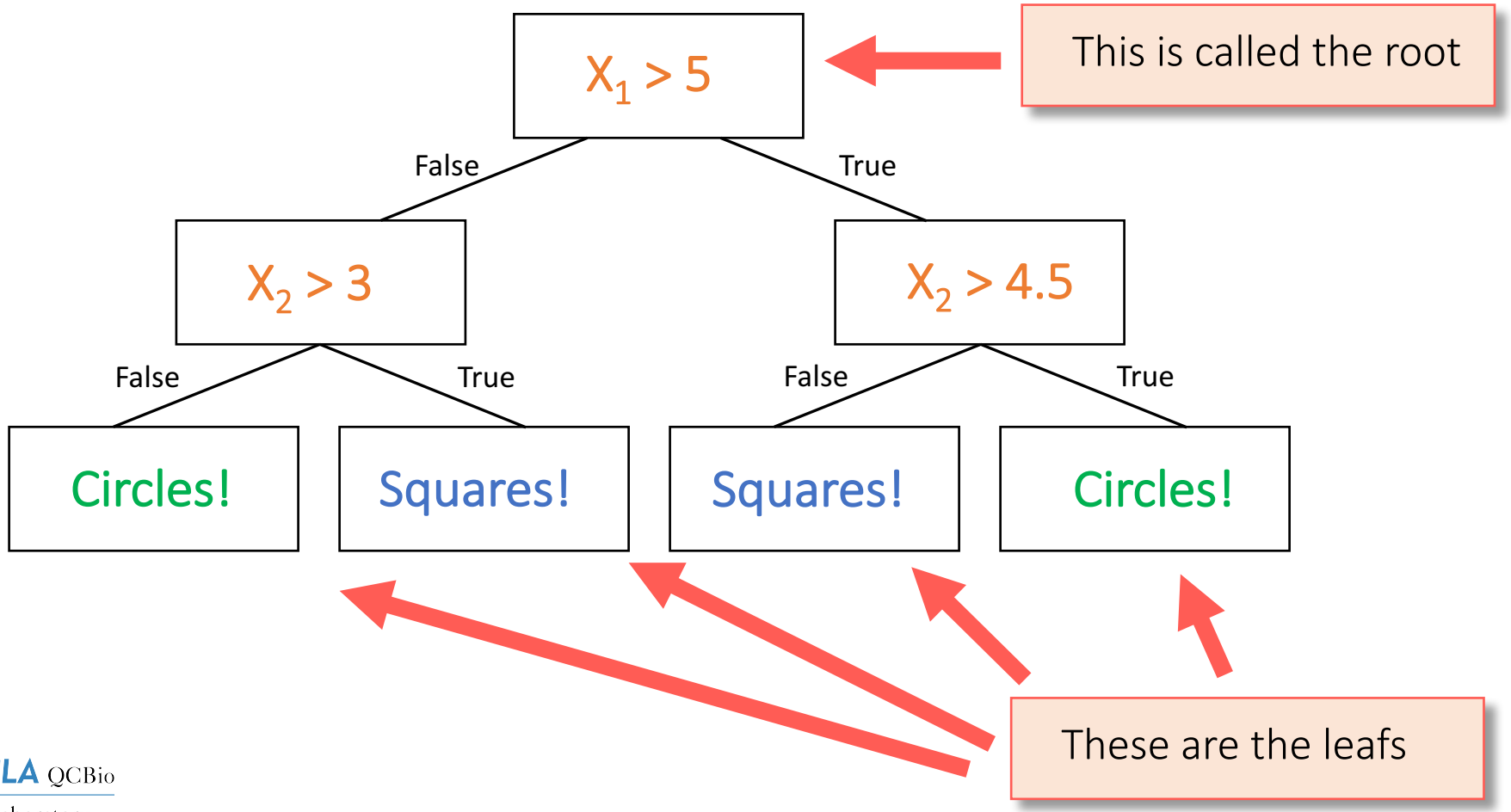


Decision tree

Photo by Shlomit Pinter
https://shlomitpinter.wixsite.com/shlomit

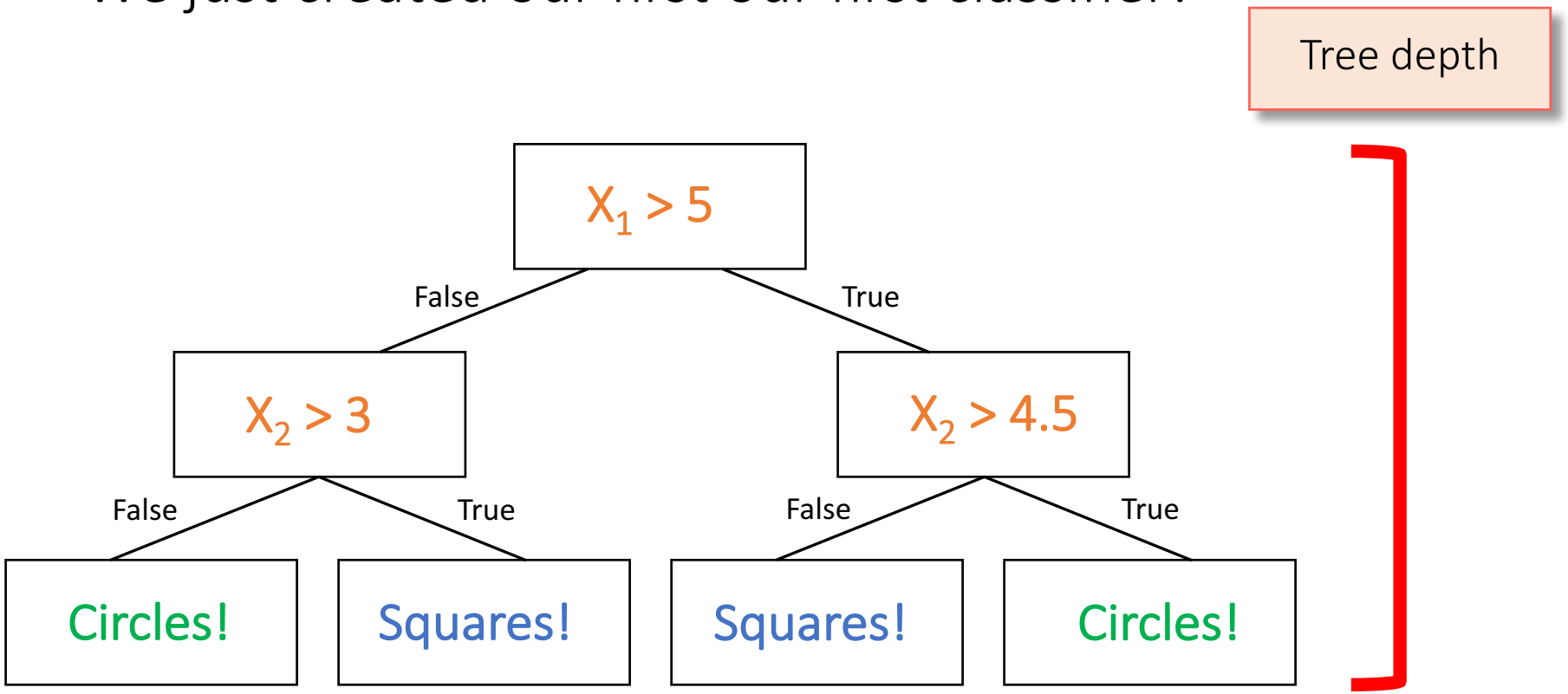
Decision tree

We just created our first our first classifier!



Decision tree

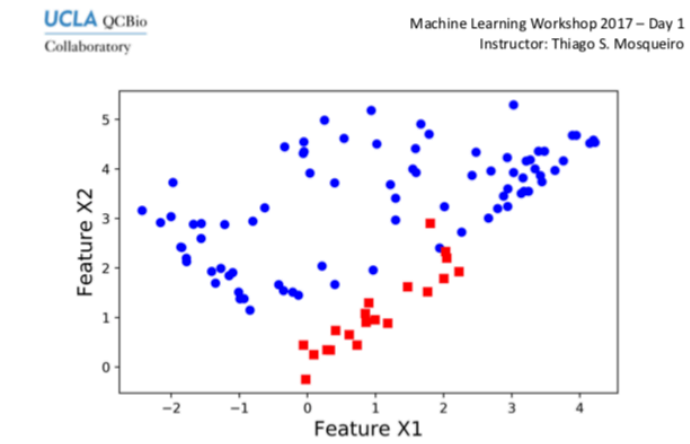
We just created our first our first classifier!

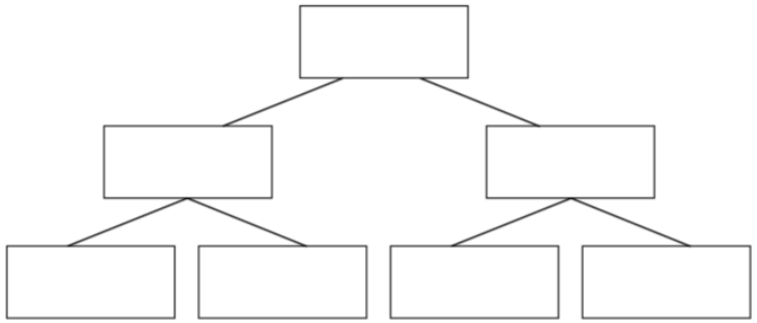


Practice – Decision trees

Let's do a quick exercise and construct our very first Decision Tree based on some data…

- You should have a paper sheet like the one on the right
- Take 5 min to construct your own decision tree
- There's no right or wrong solution!
- Try and draw each of the lines separating the dataset as we did in the previous slides!
- Fill the boxes with your inequalities
- Call one of the instructors if you are having problems





Random Forests

Among the most widely used classifiers are the Random Forests, which are composed of many decisions trees that consider only a subset of the features (and often the data samples).

Now that everybody has their own Decision Tree,

the class just became a giant Random Forest!

To find the prediction of this Random Forest, we make a poll and consider the class with most votes

Quick test

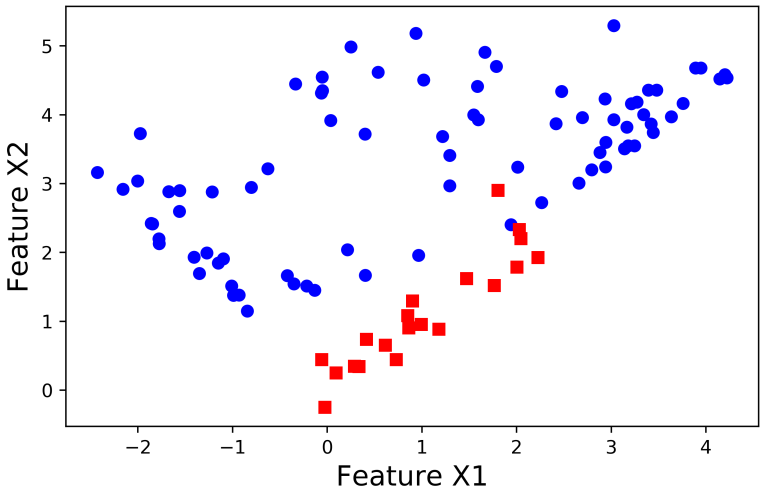
In the previous exercise, we just created a Random Forest for the dataset in the plots.

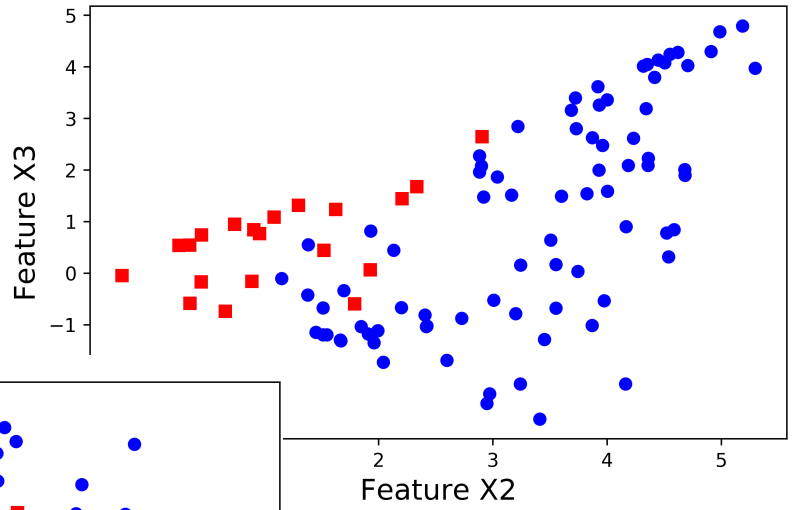
Let's see our prediction, as a group, for:

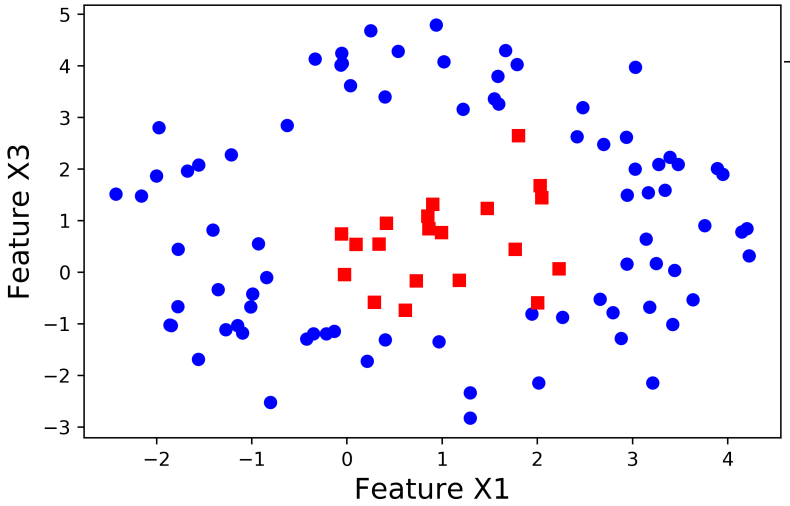
X1 = 0

X2 = 2

X3 = 0







Classifier

Essentially, this is how a classifier is constructed

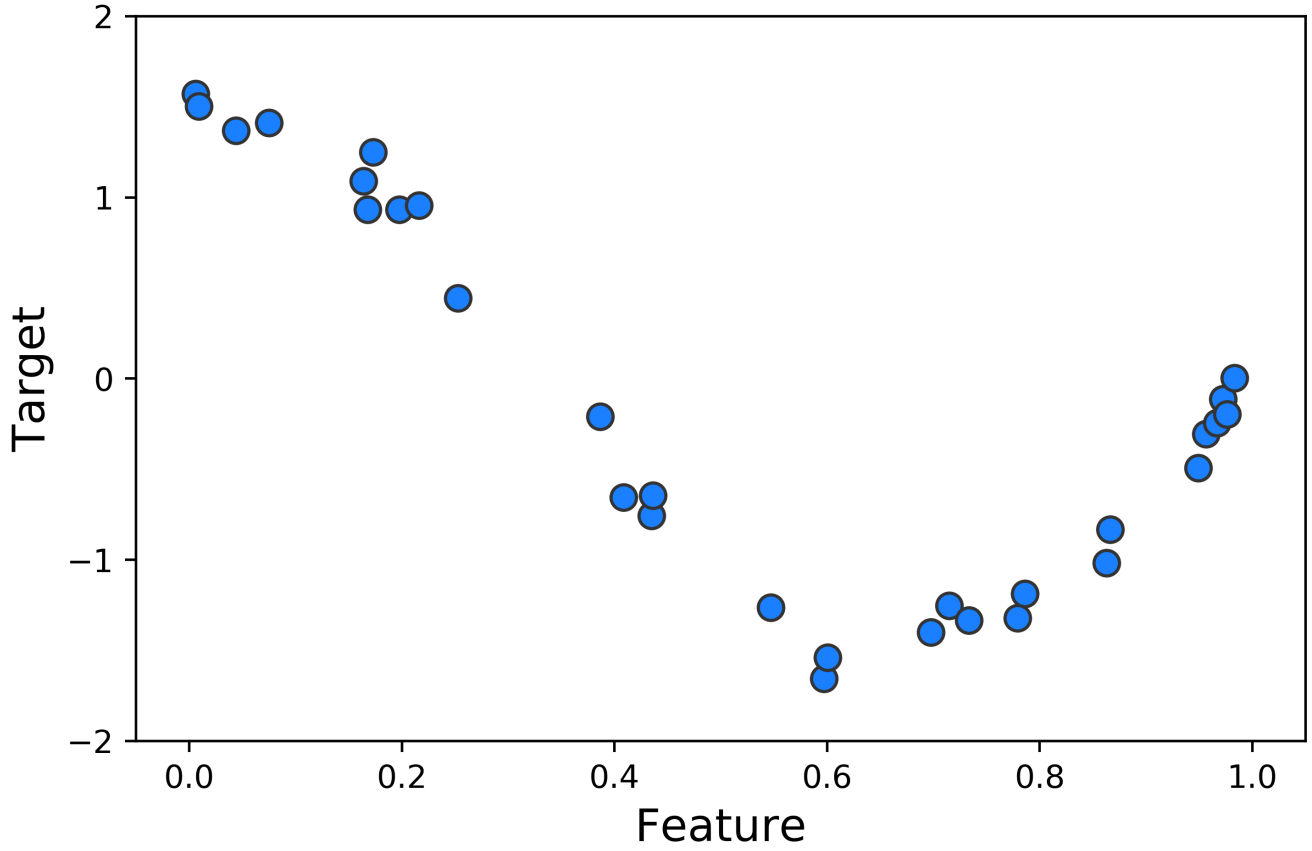
Different models have different algorithms, but the general idea is often based on some optimization procedure.

The bottom line is: A classifier is simply a function that tells you to which class a point belongs to

Tomorrow we will explore in depth how to create our models using Scikit-Learn with very few lines of code!

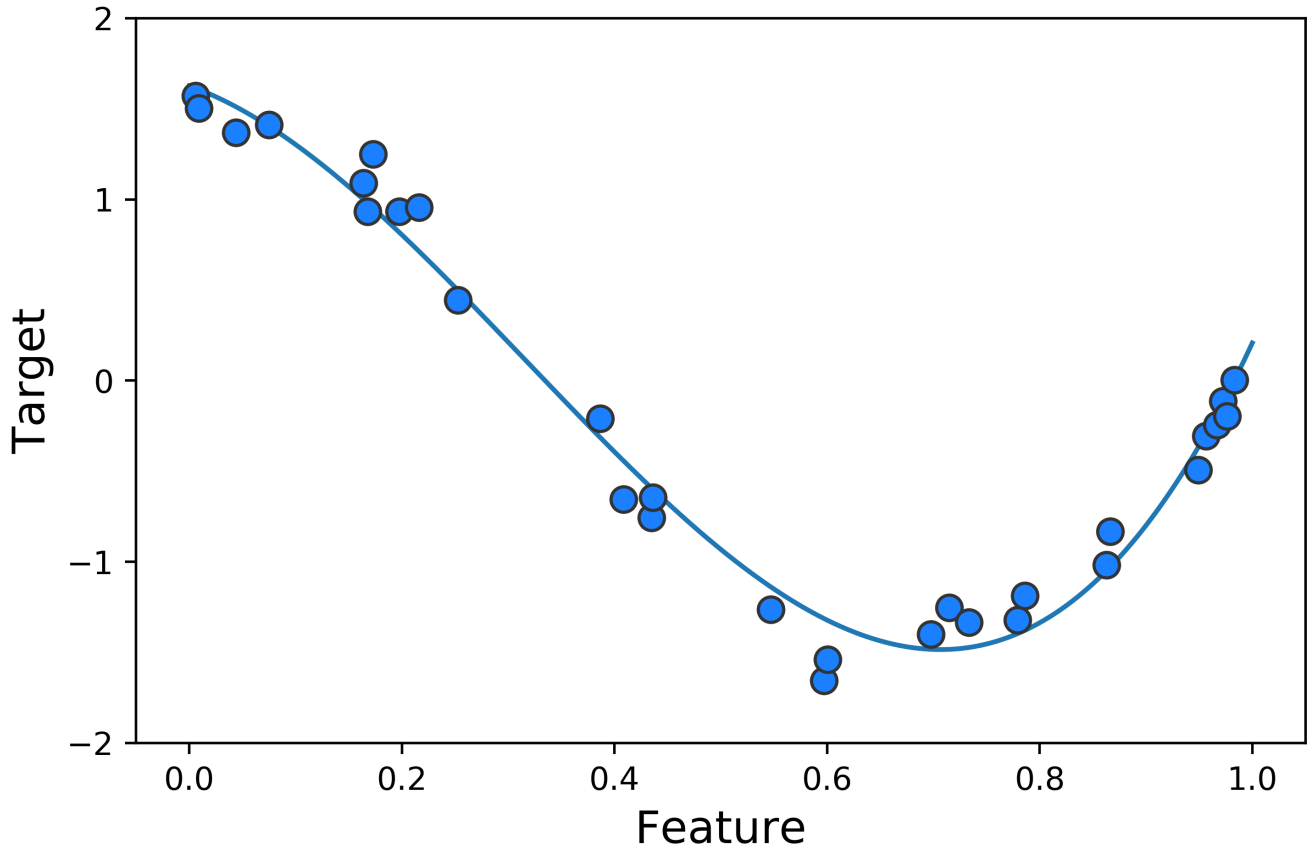
How to choose a model?

Suppose you are trying to perform a regression using the data below. Which model would you use?



Ideal solution

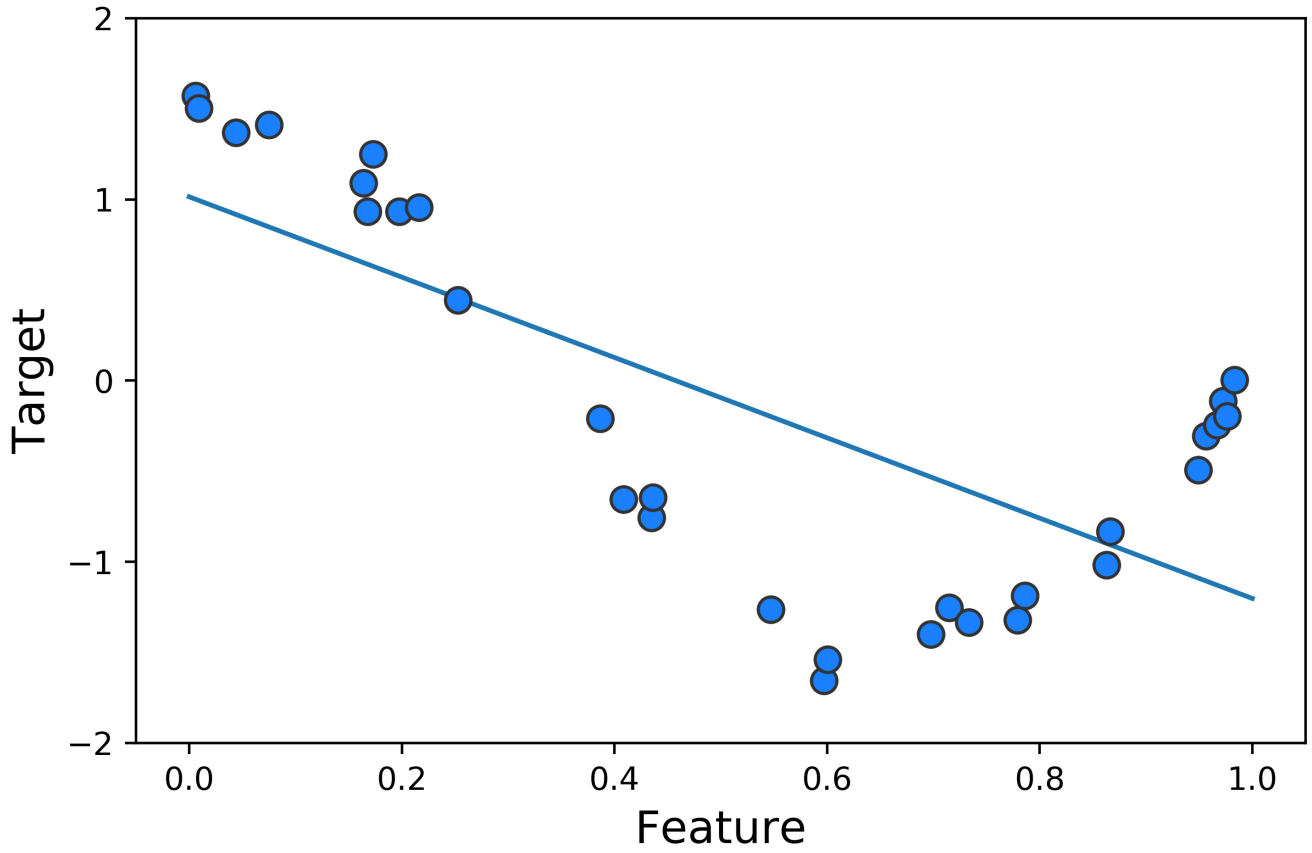
The ideal solution should look somewhat like this.



Underfitting

If you model behaves like this,

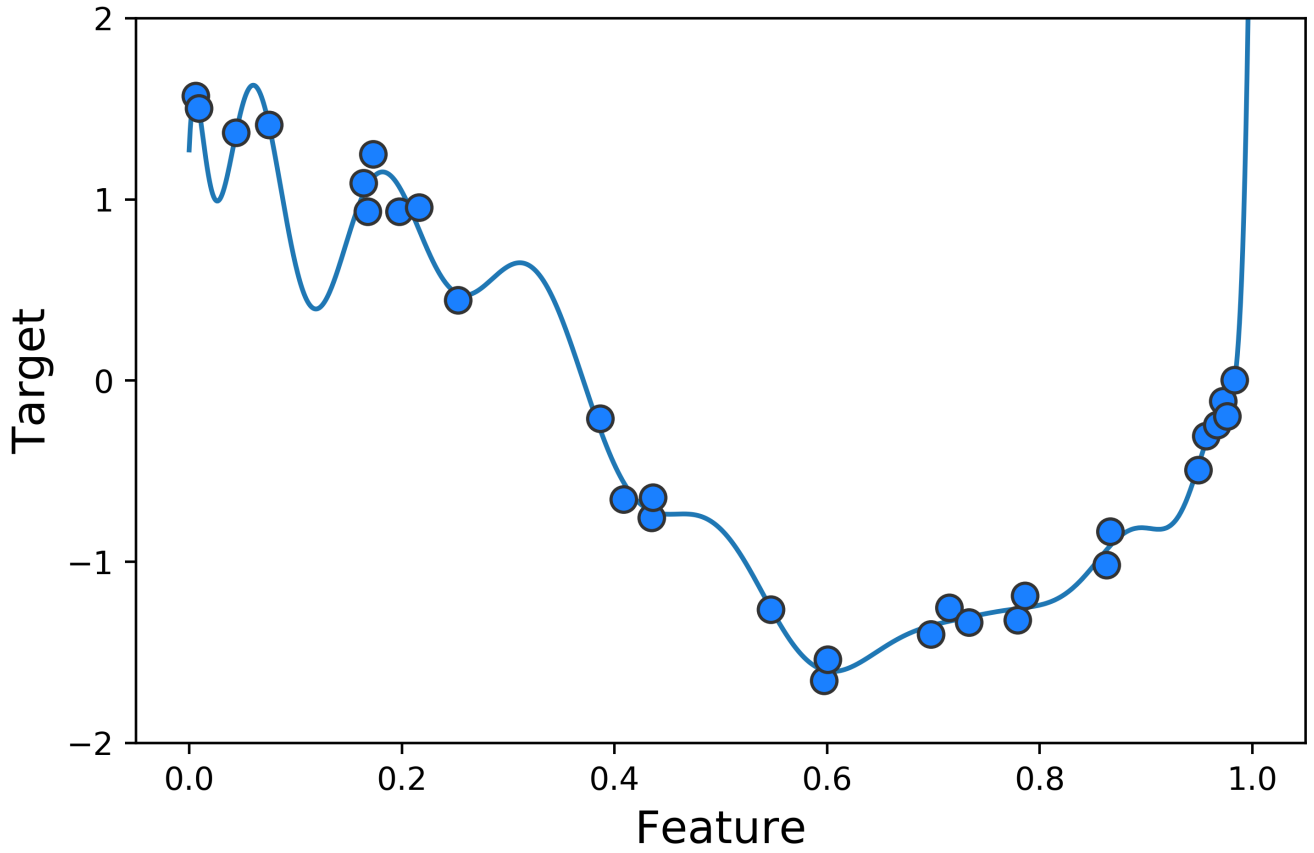
there is something missing with it…



Overfitting

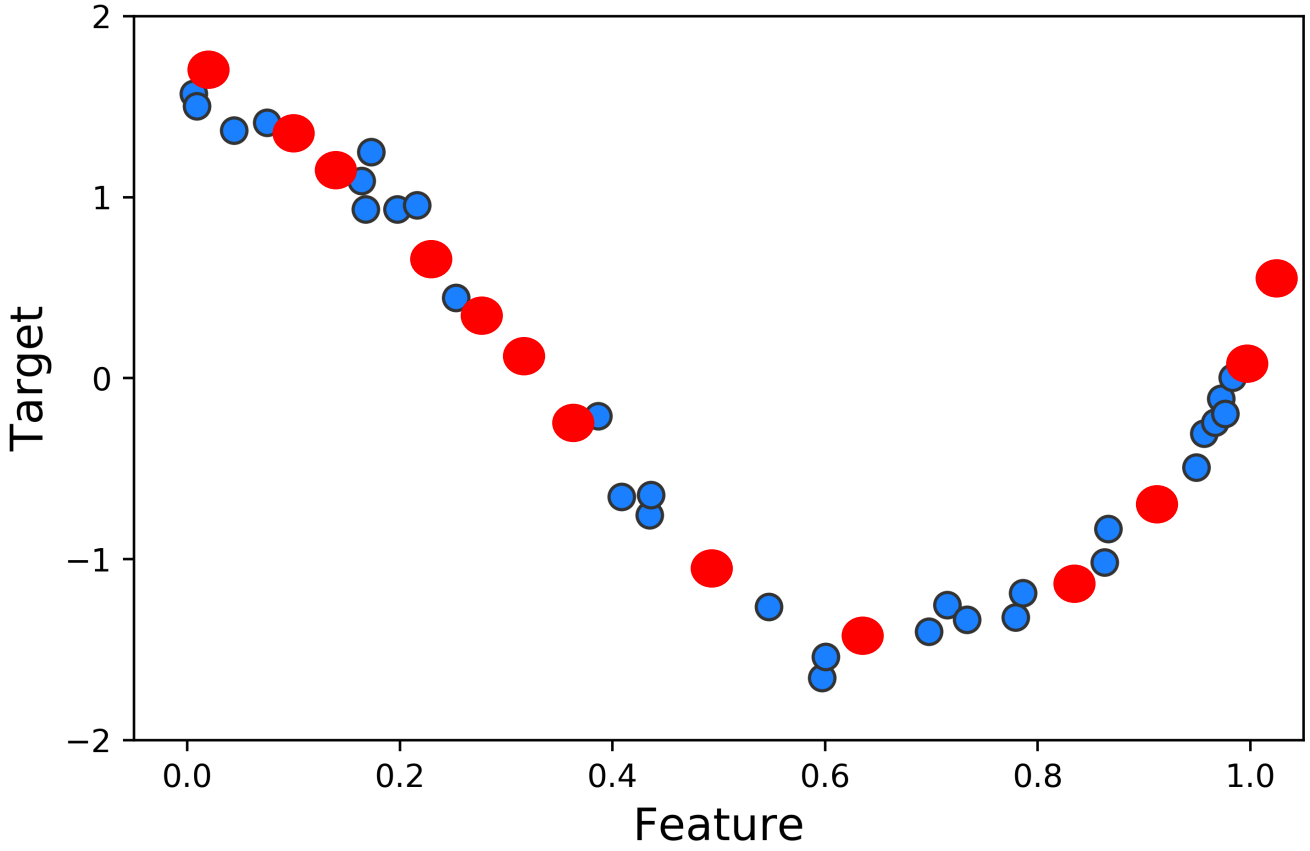
On the other hand, your model may be

more complicated or complex than the data requires…



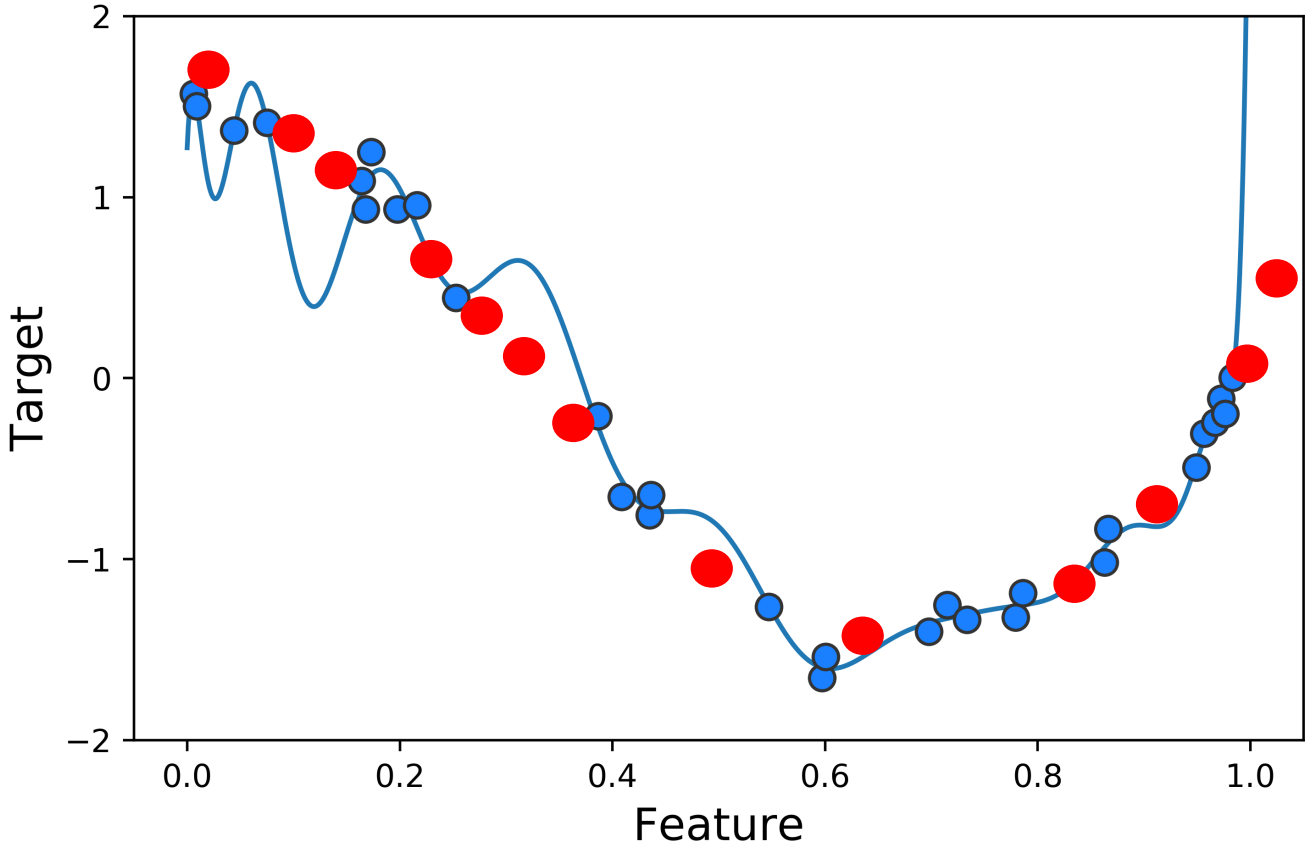
Generalization

Let's say that new data was collected, represented by the red circles below



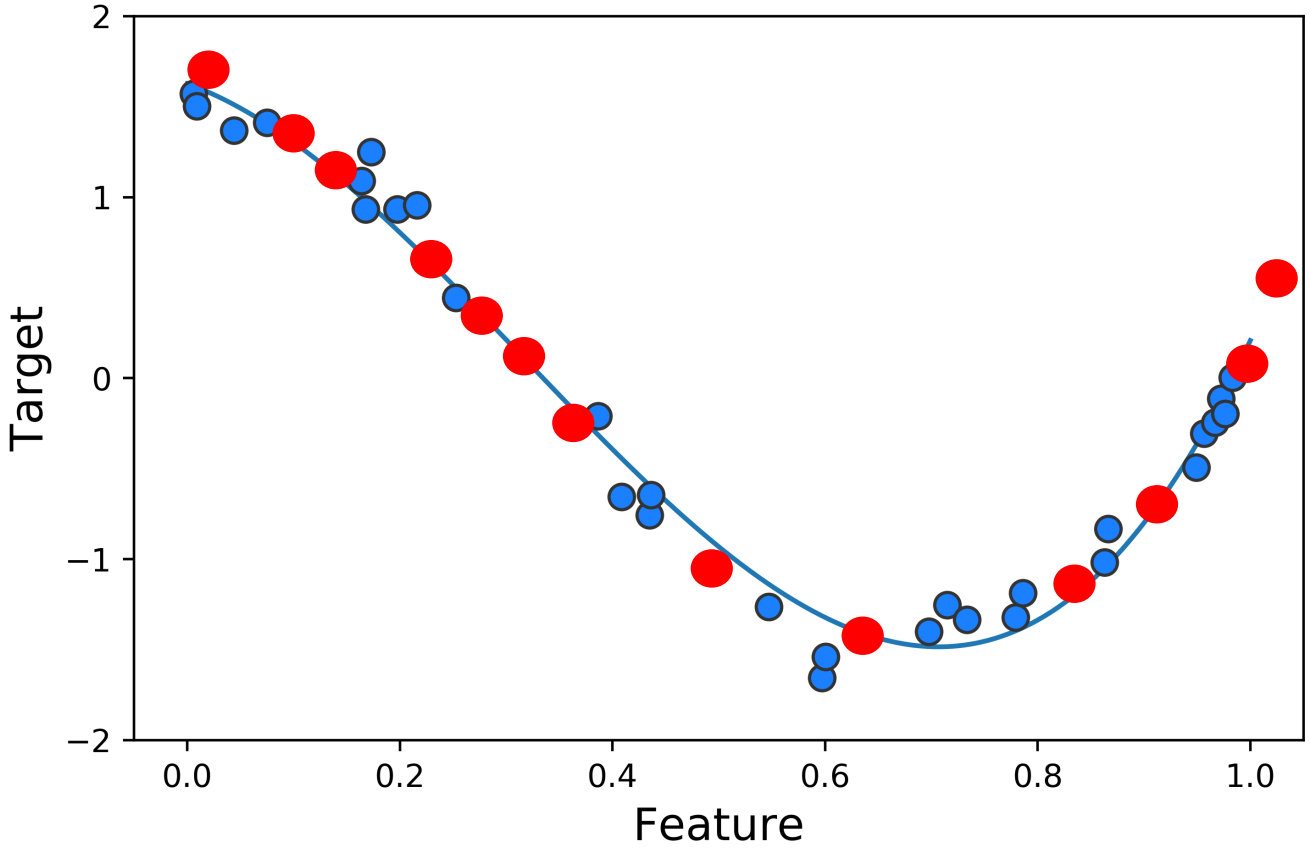
Generalization

Although the overfitted solution perfeclty described the training data, it fails the new data!



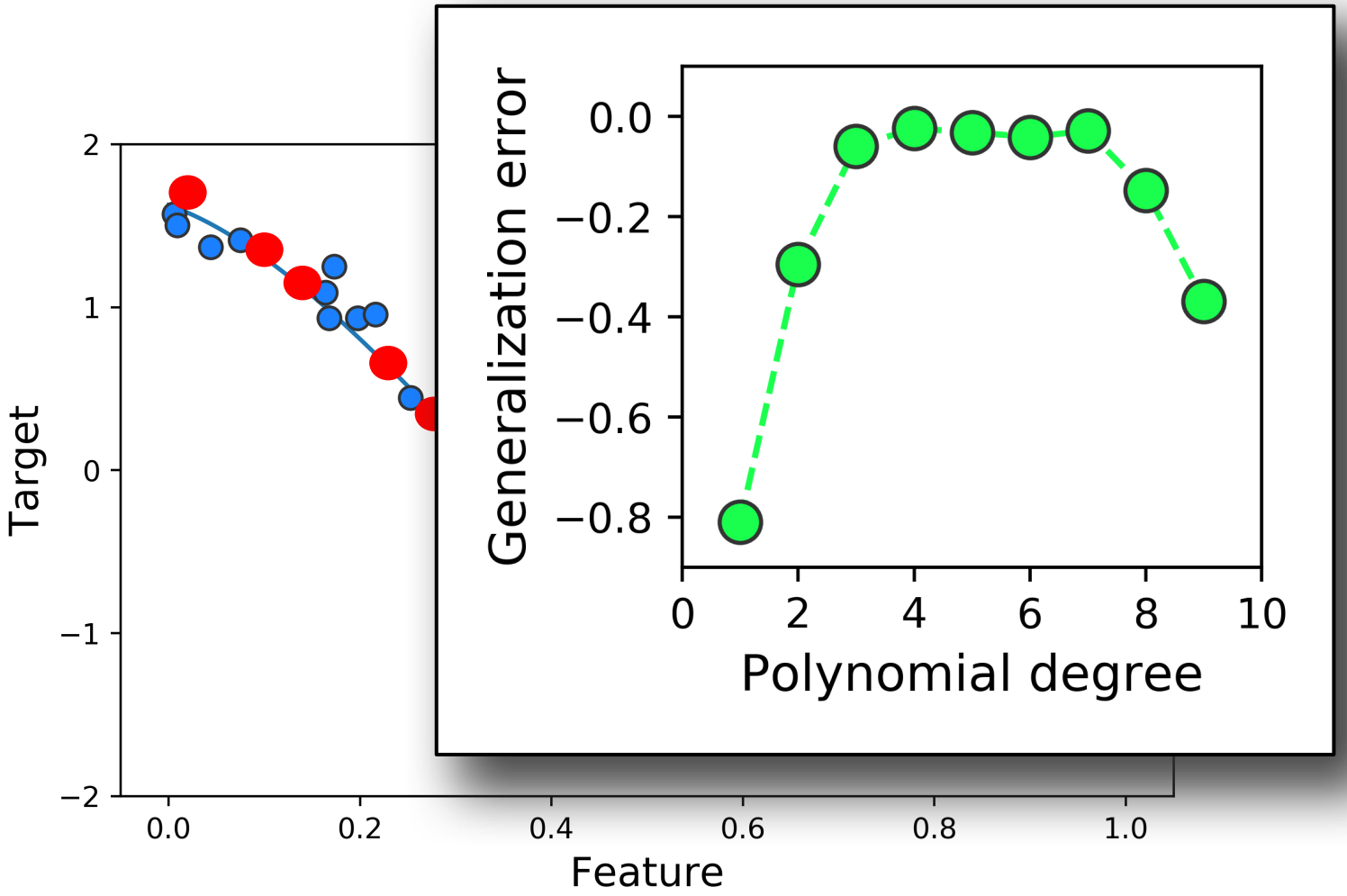
Generalization

A the ideal solution remains with a reasonable performance!



Generalization

A the ideal solution remains with a reasonable performance!

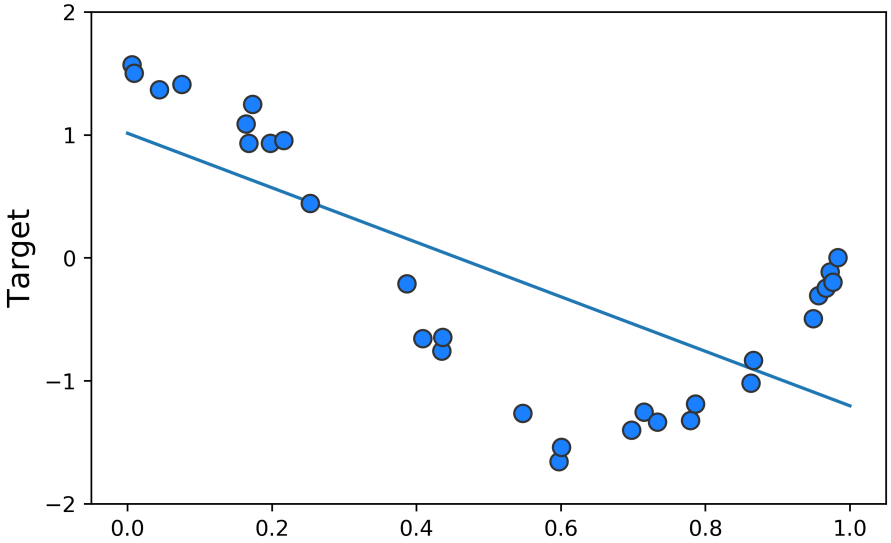


Overfitting & Underfitting

Underfitting occurs when a model cannot adequately capture the underlying structure of the data.

How do you solve it?

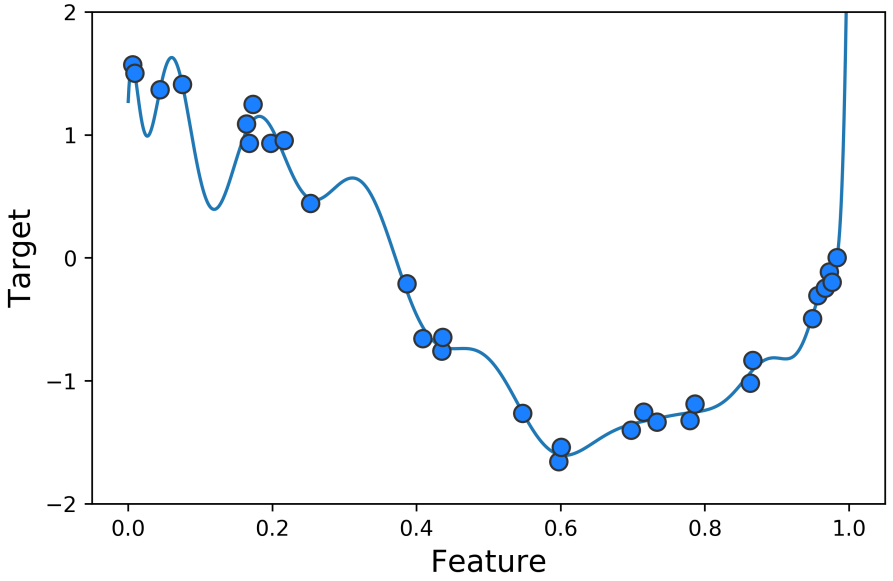
Choose another model, something with higher complexity



Overfitting occurs when a model reproduces exactly a particular set of data, and likely fails to explain new data points or predict future observations.

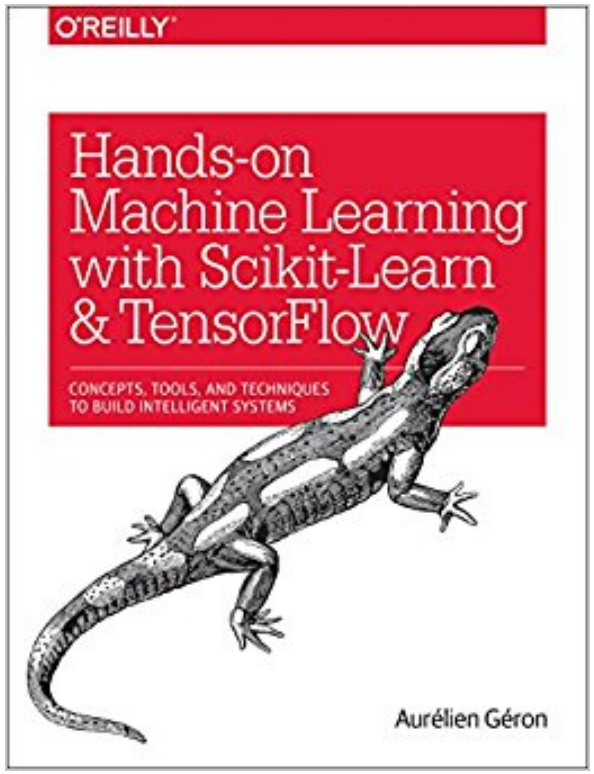
How do you solve it?

Model selection and cross-validation

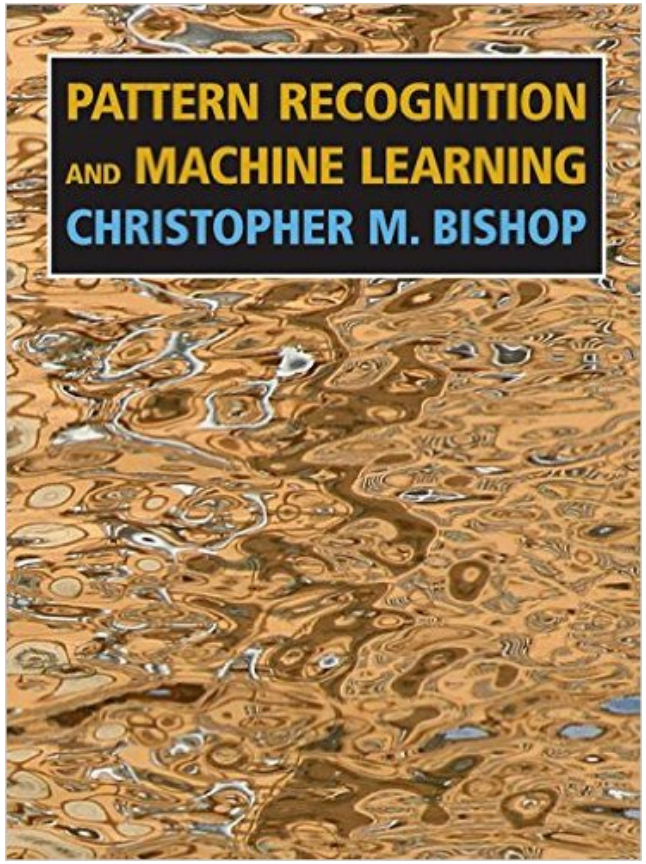


For more resources…

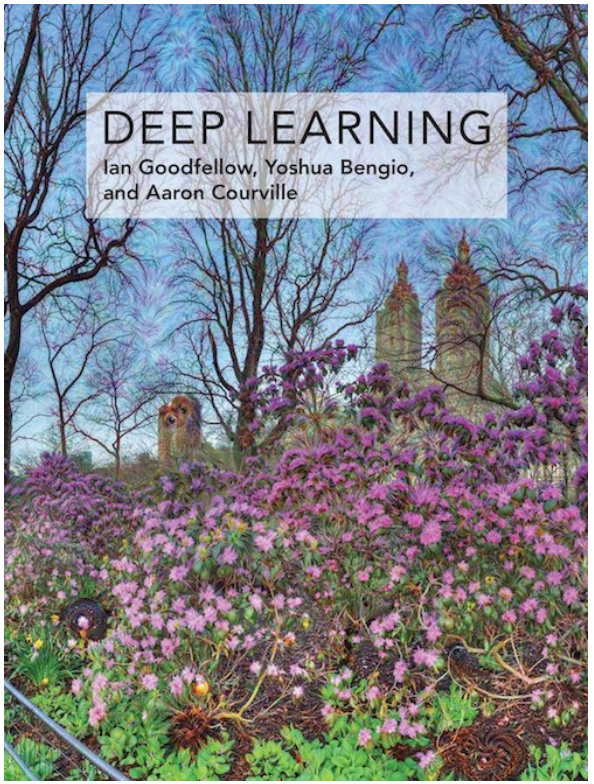
Hands-on Machine Learning
Aurélien Géron



Pattern Recognition and Machine Learning
Christopher Bishop



Deep Learning
Ian Goodfellow et a.



Getting start with Scikit-Learn

Let's use PCA from Scikit-Learn

To give you an idea of how Scikit-Learn is simple to use, let's perform a PCA on a dataset

This loads the Breast Cancer Wisconsin Dataset

Sample features

Target for prediction

In [27]:
```
import sklearn
import pylab as pl

import sklearn.datasets
bcancer = sklearn.datasets.load_breast_cancer()
```

In [28]:
```
print("Features: ", bcancer.data.shape)
print("Labels: ", bcancer.target.shape)
```

```
Features:  (569, 30)
Labels:  (569,)
```

Let's use PCA from Scikit-Learn

Look at the target variable:

```
In [29]: print( bcancer.target )
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

It identifies which entries were indeed positive for cancer

Running a PCA

Let's google "PCA scikit learn"

sklearn.decomposition.PCA

class `sklearn.decomposition.` **PCA** (*n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None*) [source]

Principal component analysis (PCA)

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

Notice that this class does not support sparse input. See `TruncatedSVD` for an alternative with sparse data.

Read more in the User Guide.

Parameters: **n_components** : int, float, None or string

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

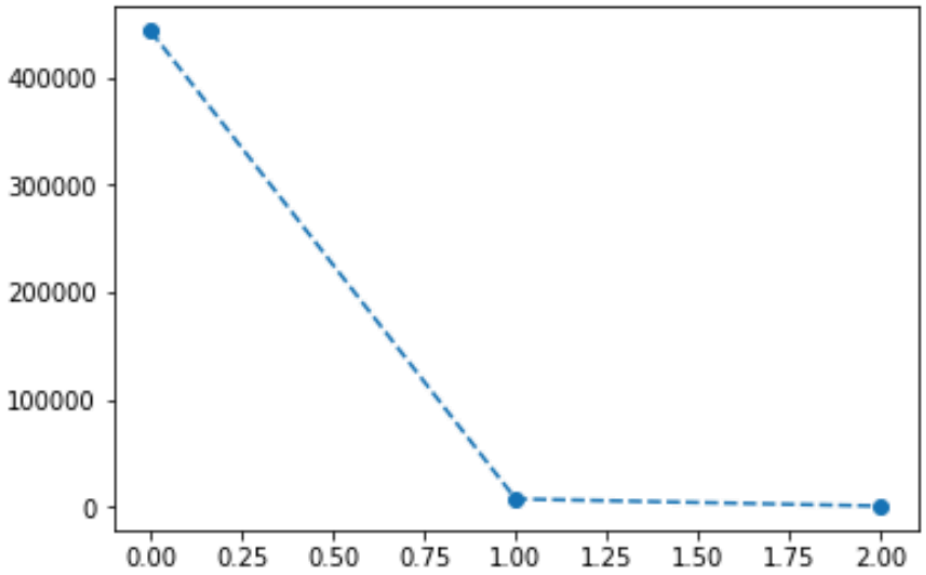
if n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension if `0 < n_components < 1` and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than

Running a PCA

In [33]:
```
from sklearn.decomposition import PCA

bcancer_pca = PCA( n_components=3 )
bcancer_pca.fit(bcancer.data)
```

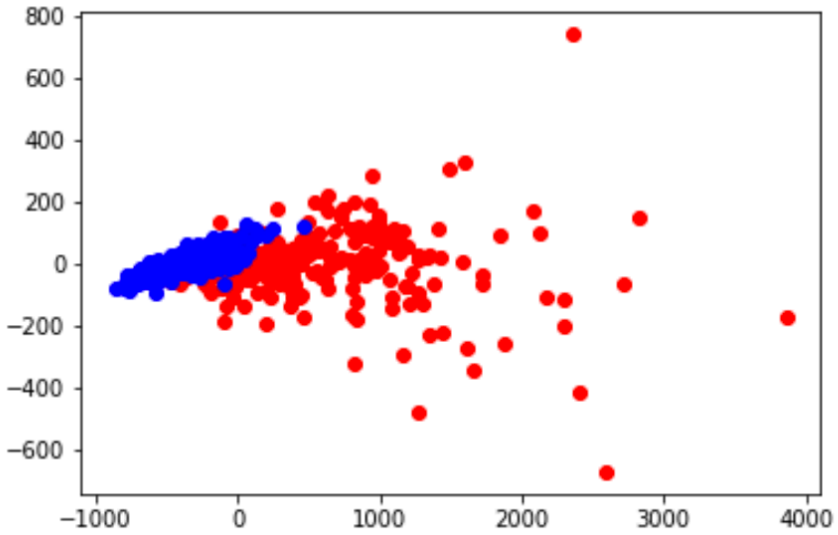
Out[33]:
```
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [34]:
```
pl.plot( bcancer_pca.explained_variance_, 'o--' )
pl.show()
```



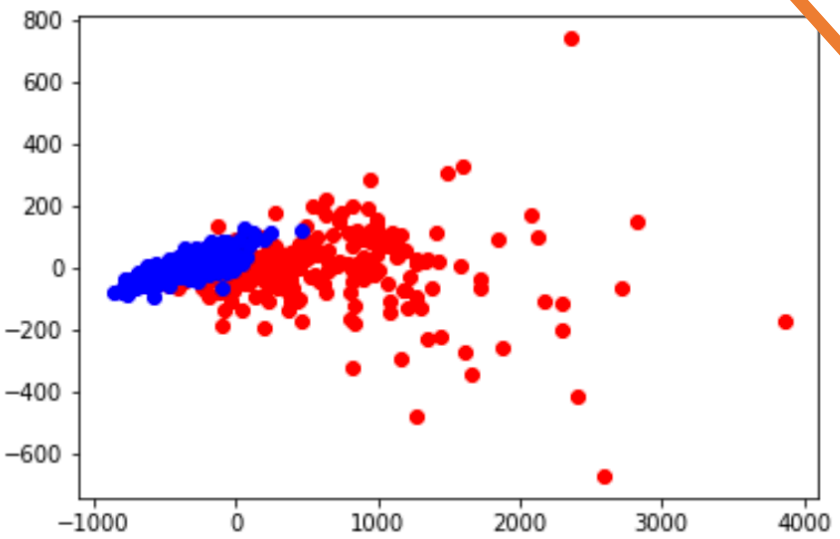
Let's plot the first two components

In [35]:
```
index1 = bcancer.target == 1
index0 = bcancer.target == 0

transfFeatures = bcancer_pca.transform( bcancer.data )    # Finding the Principal Components

pl.plot( transfFeatures[index0,0], transfFeatures[index0,1], 'ro' )
pl.plot( transfFeatures[index1,0], transfFeatures[index1,1], 'bo' )
pl.show()
```



Let's plot the first two components

```
In [35]: index1 = bcancer.target == 1
         index0 = bcancer.target == 0

         transfFeatures = bcancer_pca.transform( bcancer.data )    # Finding the Principal Components

         pl.plot( transfFeatures[index0,0], transfFeatures[index0,1], 'ro' )
         pl.plot( transfFeatures[index1,0], transfFeatures[index1,1], 'bo' )
         pl.show()
```



This serves to select entries according to their target value.

But wait, how does it work??

Logical operators

In python we can use logical operators to compare two numbers

We can do the same with NumPy arrays!

```
In [16]: print( 1 == 1 )
         print( 1 > 1 )
         print( 1 >= 1 )
```

```
True
False
True
```

These operators test each element of the array

```
In [17]: print( b > 3 )
         print( c > 3 )
```

```
[False False False  True  True]
[ True  True False False False]
```

It gives you True's only when the condition is satisfied

We can use these operators to select a particular set of elements:

```
In [18]: print( b[b > 3] )
         print( b[c > 3] )
```

```
[4 5]
[1 2]
```

Logical operators

In python we can use logical operators to compare two numbers

We can do the same with NumPy arrays!

```
In [16]: print( 1 == 1 )
         print( 1 > 1 )
         print( 1 >= 1 )
```

```
True
False
True
```

These operators test each element of the array

```
In [17]: print( b > 3 )
         print( c > 3 )
```

```
[False False False  True  True]
[ True  True False False False]
```

It gives you True's only when the condition is satisfied

We can use these operators to select a particu

```
In [18]: print( b[b > 3] )
         print( b[c > 3] )
```

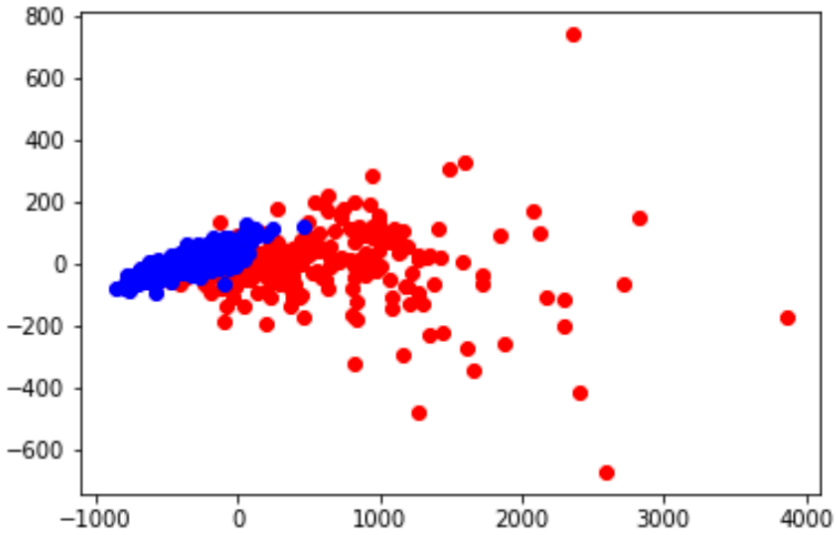
```
[4 5]
[1 2]
```

Select elements of b in which b > 3

Select elements of b in which c > 3

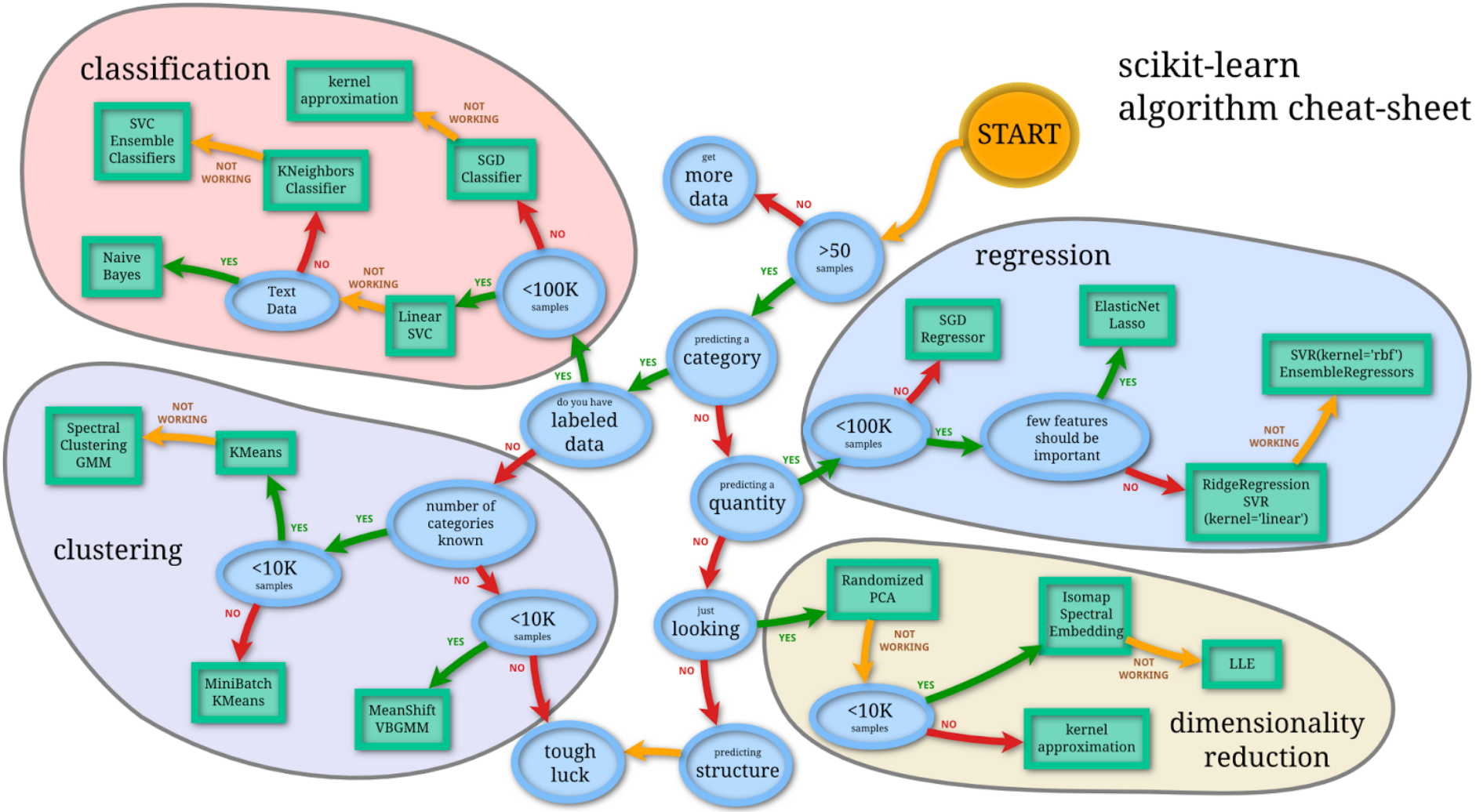
Let's plot the first two components

In [35]:
```
index1 = bcancer.target == 1
index0 = bcancer.target == 0

transfFeatures = bcancer_pca.transform( bcancer.data )    # Finding the Principal Components

pl.plot( transfFeatures[index0,0], transfFeatures[index0,1], 'ro' )
pl.plot( transfFeatures[index1,0], transfFeatures[index1,1], 'bo' )
pl.show()
```

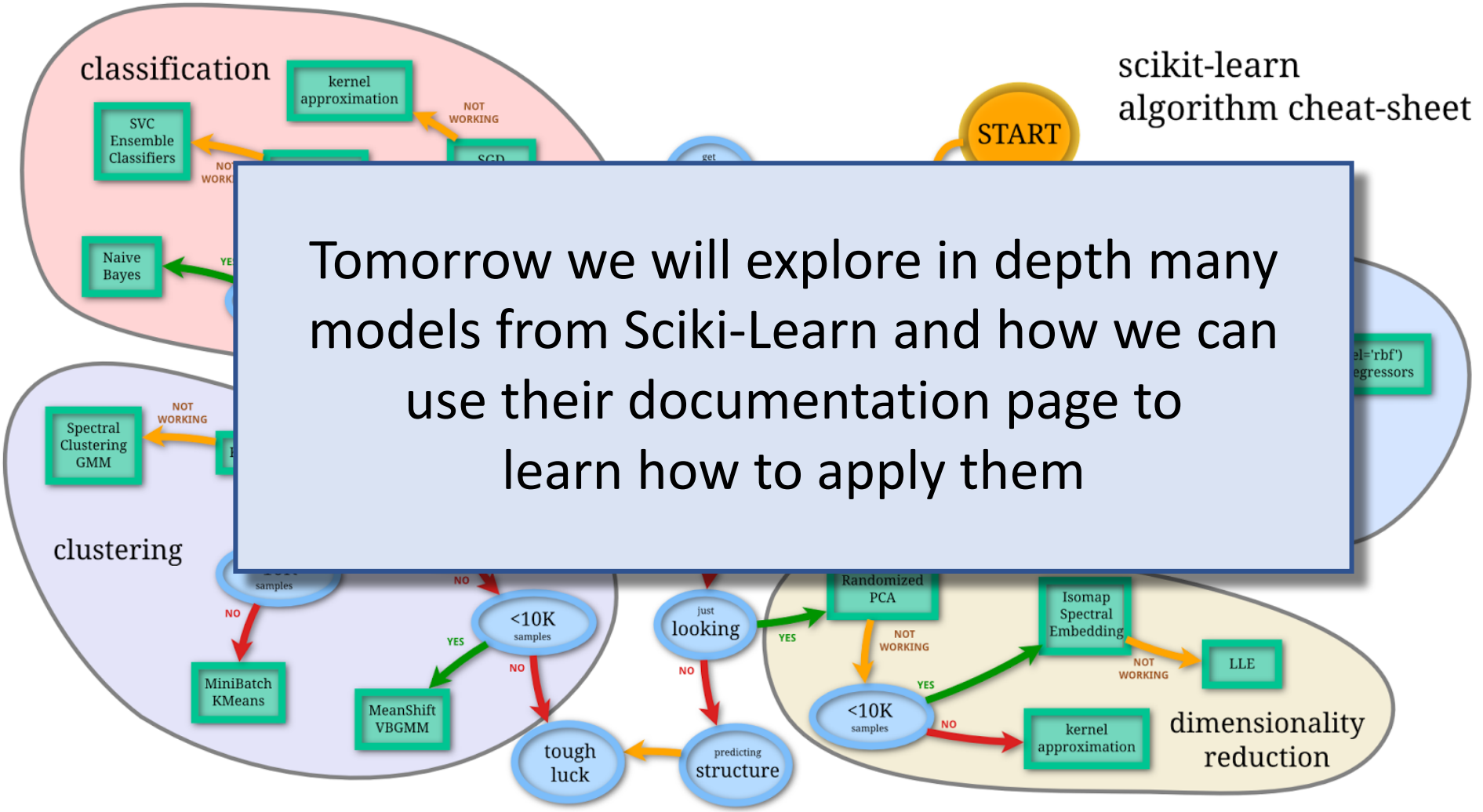


We can see a good **distinction** between points from classes 0 and 1

Scikit-learn's cheat sheet



Scikit-learn's cheat sheet



Practice

Practice 1

The easiest way to consolidate the basics of Machine Learning is to apply to your own work and context.

- Think of one example where you could use classification in one of your projects
- Think of one example where you could use regression in one of your projects
- Design one of the previous problems, stating the what is your target variables, and what features you would use to construct your model

Practice 2

With the synthetic dataset used when we were introduced to Decision Trees, write some Python code to:

- Use Matplotlib's gallery to plot this dataset in 3D
  hint: look for *scatter plot in 3D*
- Change the angle of the previous visualization
- Find the Principal Components of this dataset
- Plot the normalized eigenvalues. Anything interesting?
- Plot in 2D the first two first Principal Components, color-coding the class to which each point belongs to

Practice 3

If you want to practice Python, implement the Decision Tree that you created earlier today as a function in Python. You usually don't need to implement it directly (see Day 2 for more), but this is a good programming exercise.

- Implement each branching as an if condition
- Transform your code in a function
- Try some points and check if the result is as expected.

Practice 4

Try using a different dataset from Sciki-Learn. You can follow the slides and practice what we discussed today by yourself.

- Look for the Wine Datasets are available on Scikit-Learn
- Load the data in a new notebook
  hint: load_wine()
- How many samples and features does this dataset have? How many classes?
- Plot this dataset, color-coding the class
- Perform a PCA and re-plot it

Practice 5

Let's use the notebook that we created during this first day and learn how to share a notebook with a colleague or collaborator.

- Finish everything you want to have on your notebook. Add anything you want.
- Save and close it.
- Look for the notebook file on your hard drive. Rename it to add your own initials to it.
- Exchange notebooks with someone from this class. You can use e-mails.
- Open the notebook you received and run it.

What’s in for tomorrow…

Day 2– What are our next steps?

- We will dive in the world of Classification Models
- We will study several different models and their range of applicability
- We will also understand how to assess their performance for a given classification task
- We will learn about generalization and how to make sure our model is not overfitted
- We will explore Scikit-Learn library and their documentation pages