Sargent U Intro to ML Course

Hitarth and Andy
April 13, 2020

Course Overview

- Introduction to Linux
- Introduction to Python
- Data Curation, Formatting and Standardization with Python
- Simple machine-learning modelling for experimental acceleration
- (If interest/time) Advanced ML modelling, interfacing DFT and ML

General Course Notes

- We will upload all documents to the shared folder including:
  - Course slides
  - Working python notebooks
  - Assignments
  - Assignment solutions (maybe)
  - Other resources
- Please ask questions! We are not doing this for us. We are doing it so that everyone can learn - and that means making sure everyone keeps up.
- Exit full screen mode on Zoom (unless you have a multiple displays) so that it is easier to follow along and open/close your command terminal.

Intro To Linux

Overview:

- What is Linux?
- Why do you care?
- Accessing Linux - ssh
- Linux basic commands
- Introduction to vim
- Advanced Topics:
  - Aliases
  - Authorized Keys
  - Setting up vimrc

Schedule

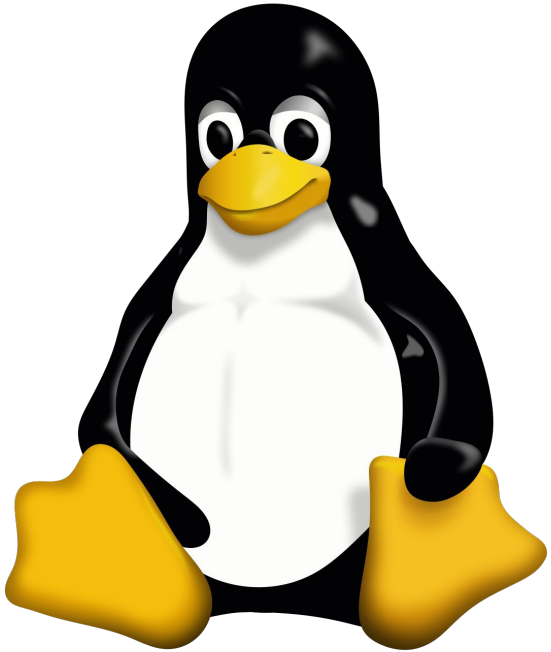
I have no idea how long this will take.

We will take a break at the “Summary so far” slide or at 10:45, whichever comes first.

We should be done early today - we can then move into some of the Python slides, but it may be better to do those in one day rather than split them up.

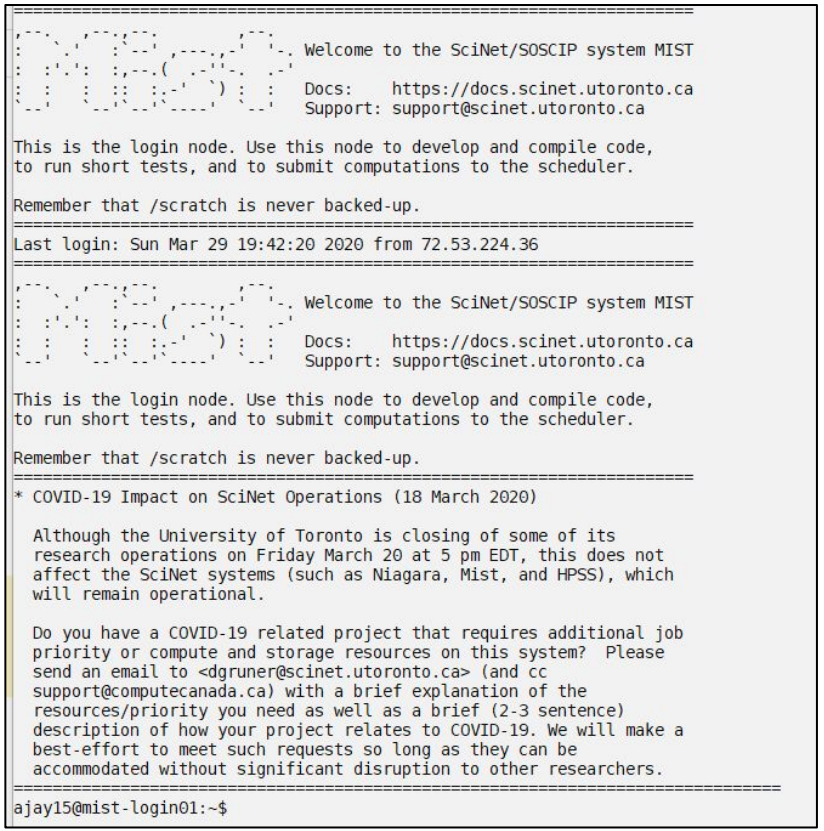
What is Linux?

- Linux is an **O**perating **S**ystem (OS), just like Windows or macOS.
- This means it manages the communication between your software and your hardware
- Why use Linux?
- It is FREE
- Open source
- More flexible than Windows or iOS
- Our supercomputing clusters are linux-based :)



Why do I care?

- If you want to use our supercomputing clusters, you NEED to be familiar with Linux.
- This means that everyone who wants some DFT for their Nature paper needs to be familiar with basic commands.
- GPU-based machine learning (ML) also requires linux; we won't be using supercomputers in this course, but next week in the DFT courses you will!



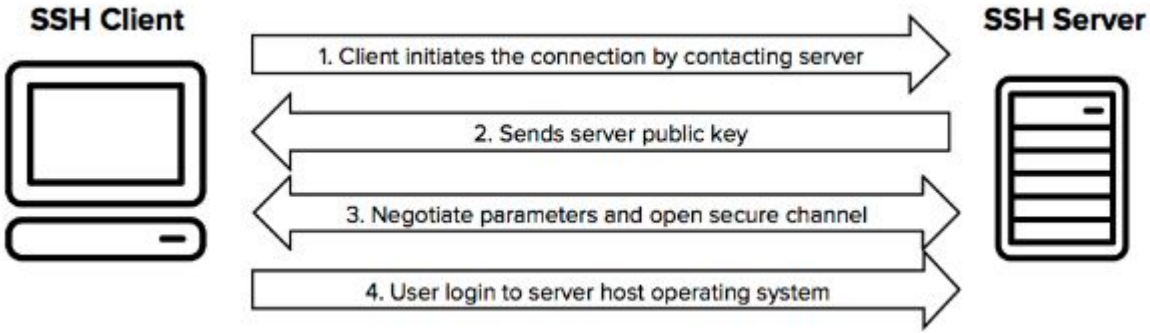
Linux Navigations - Some Notes

- We will walk you through different commands in Linux.
- When you see something surrounded by pointy brackets, i.e <USERNAME> it means you need to replace whatever is between the brackets with something specific (like a name, etc). Also remove the brackets!
- If you have any questions - interrupt us and ask, or raise a hand in the chat, or message the chat, or message Andy/Hitarth privately, and we can help you.
- This is meant to be interactive! We want you following along with what we are doing - if you get behind it may be tough to catch up, so tell us if we are going too fast!
- Finally, you can find a great resource here: https://ryanstutorials.net/linuxtutorial

Logging into a linux server - SSH

- Linux servers (or, computers), can be accessed remotely using **S**ecure **Sh**ell (ssh) connections.
- An SSH connection uses encryption to ensure that all authentication, commands and data transfers are encrypted against outside attacks.
- This allows users to log in to a server remotely, from their own personal computer, without fear of data being stolen
- Think of this kind of like TeamViewer, but for Linux

SSH - Visually



Your computer

Supercomputer cluster

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

SSH - logging in from Mac/Windows 10

If you responded to the email about this workshop, you should have an account on our **local sargent ML server**.
If you are on Mac or Windows 10, log in by opening a terminal or command prompt, and typing in the following:

ssh <USERNAME>@142.1.178.186

- Ssh - the command telling your machine to open a connection
- <Username> - your userID on the server (can be found on the Google Docs where you signed up)
- Host name - The IP address of the server to which you are logging in

You will be prompted for authorization (say yes - twice) and a password, which has been set to: **sargent_lab.** Type it in and hit enter (don't worry that you can't see anything being typed in!)

SSH - logging in from Windows < 10

Before recent Windows 10 updates, Windows did not support SSH functionality natively

We use a third - party software, **Putty**, to ssh into linux servers to account for this.

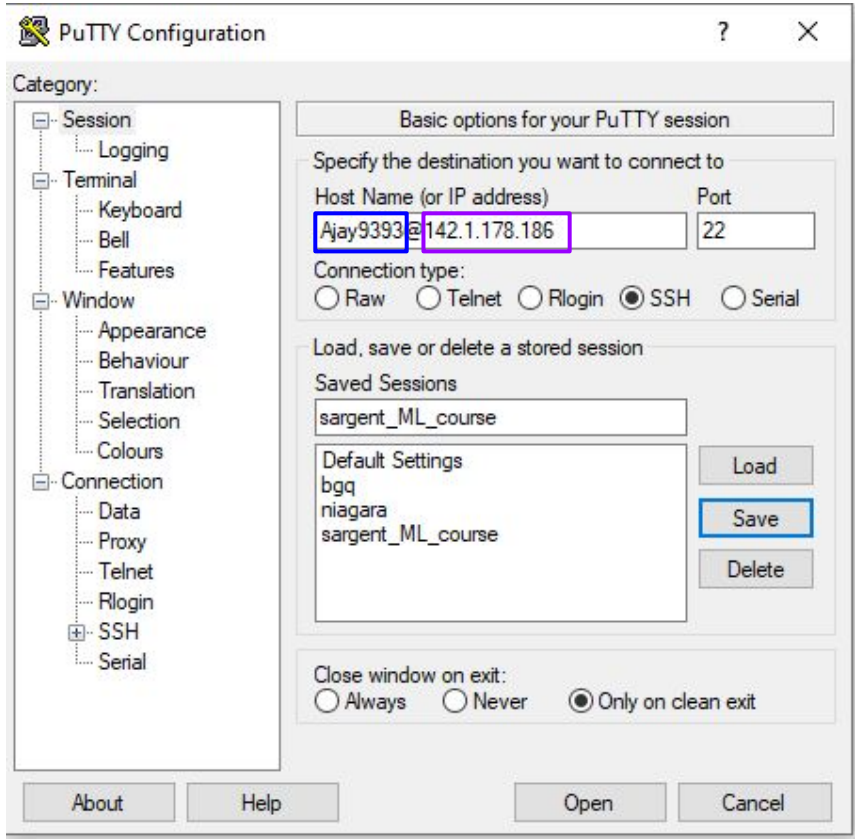
You should have downloaded Putty before this workshop, if you have not you can download it from: https://www.putty.org/

Using Putty

Enter username and host name under the Host Name (or IP address) label. Click “Open”

Should open a new terminal and ask for a password - type ***sargent_lab*** and hit enter

If you don’t want to type the command in every time, you can save the session: type a name under “Save Sessions” and click “Save”. You can then load this session and use it in the future!



Linux Navigation - basic commands

- When you log into a server, by default you log into the “home” or “root” folder.
- In Linux, you execute commands on files from the *command line*. This can be thought of as *replacing the double click* that is used in Windows and Mac.
- This means that the command - something like creating a new folder or file, opening a file with a program, running an executable, etc - must be able to see the file on which it wants to run!
- As a result, being able to easily navigate through folders on Linux is really important
- To try and make this more intuitive, for each Linux command introduced, we will display how that command is implemented in Windows

Terminology for upcoming slides

Syntax: This is the command you will type into your linux terminal. Remember, if you see <THING>, THING needs to be replaced by a name for you

Example: This will be what I am (and you should be!) typing into my (your) terminal. If required, underneath this will be the output of the command

Again - if we are going too fast, or you have questions, tell us

Where are we? The *pwd* command

Linux command

The command "*pwd*" stands for *print working directory*. It will output a string that lists the current directory.

Syntax:

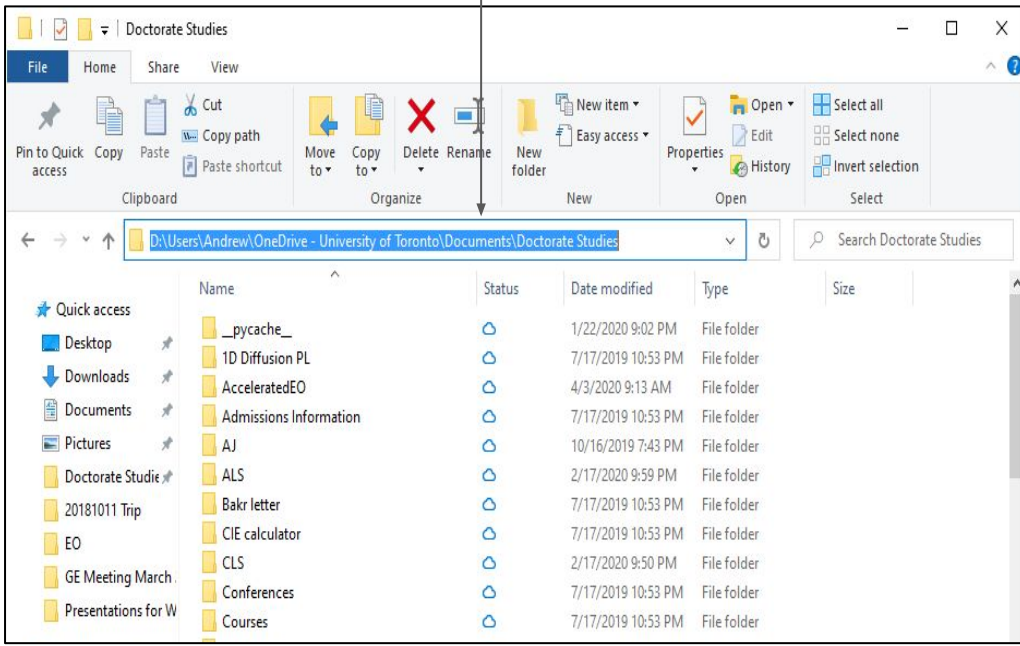
pwd

Example:

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$
```

Returns the current directory, which is Ajay9393 in the home folder.

In Windows, this is equivalent to clicking the text box (highlighted in blue) to show the folder system!



What else is here? The *ls* command

The command "*ls*" stands for *list*. It will output the documents and folders in the current directory

Syntax:

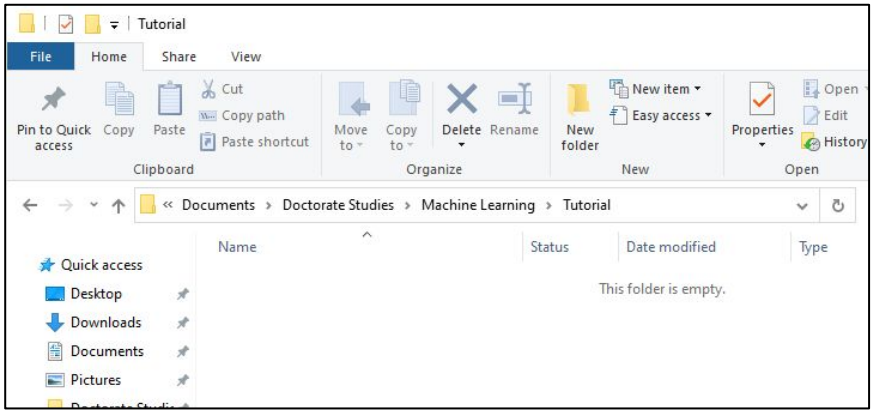
ls

Example:

```
Ajay9393@sargent_server:~$ ls
auto_ML  examples.desktop  nohup.out  test.py
Ajay9393@sargent_server:~$
```

Returns the list of files and folders in the directory Ajay9393

In Windows, this is equivalent to.. Viewing the files and folders in a directory



Making *ls* look pretty: [options]

Linux command

Command Options

The command "*ls*" stands for *list*. It will output the documents and folders in the current directory

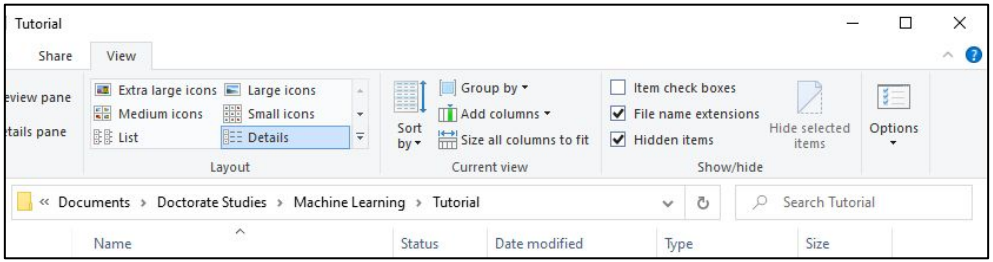
Syntax:

ls [options]

Example:

```
Ajay9393@sargent_server:~$ ls -l -t -r --color=auto
total 24
-rw-r--r-- 1 Ajay9393 Ajay9393 8980 Apr 16  2018 examples.desktop
-rw------- 1 Ajay9393 Ajay9393 2176 Sep  7  2018 nohup.out
drwx--x--x 4 Ajay9393 root     4096 Sep 27  2018 auto_ML
-rw-rw-r-- 1 Ajay9393 Ajay9393   23 Apr  3 00:06 test.py
Ajay9393@sargent_server:~$
```

In Windows, this is equivalent to modifying the view options in the directory



What did each of those -option commands do?

-l: list with long format - show permissions

-t: sort by time and date

-r: reverse order (i.e newest at bottom)

--color: colored list is automatic

There are more options! For ls, visit: ls command in Linux/Unix | list files/directories

A note on options

Typically passed as -[letter], *after* a **command**

Can be combined in simple cases, i.e:

```
Ajay9393@sargent_server:~$ ls -l -t -r --color=auto
total 24
-rw-r--r-- 1 Ajay9393 Ajay9393 8980 Apr 16  2018 examples.desktop
-rw------- 1 Ajay9393 Ajay9393 2176 Sep  7  2018 nohup.out
drwx--x--x 4 Ajay9393 root     4096 Sep 27  2018 auto_ML
-rw-rw-r-- 1 Ajay9393 Ajay9393   23 Apr  3 00:06 test.py
Ajay9393@sargent_server:~$
```

```
Ajay9393@sargent_server:~$ ls -ltr --color=auto
total 24
-rw-r--r-- 1 Ajay9393 Ajay9393 8980 Apr 16  2018 examples.desktop
-rw------- 1 Ajay9393 Ajay9393 2176 Sep  7  2018 nohup.out
drwx--x--x 4 Ajay9393 root     4096 Sep 27  2018 auto_ML
-rw-rw-r-- 1 Ajay9393 Ajay9393   23 Apr  3 00:06 test.py
Ajay9393@sargent_server:~$
```

If spelling out the full command (rather than the letter), use a double dash (i.e *--color*

Be careful to not add errant spaces!

As we move through additional commands, we will highlight useful options - but we encourage you to go play with them on your own!

Creating new folders: the *mkdir* command

Linux command

File or Directory being operated on by command

Syntax:

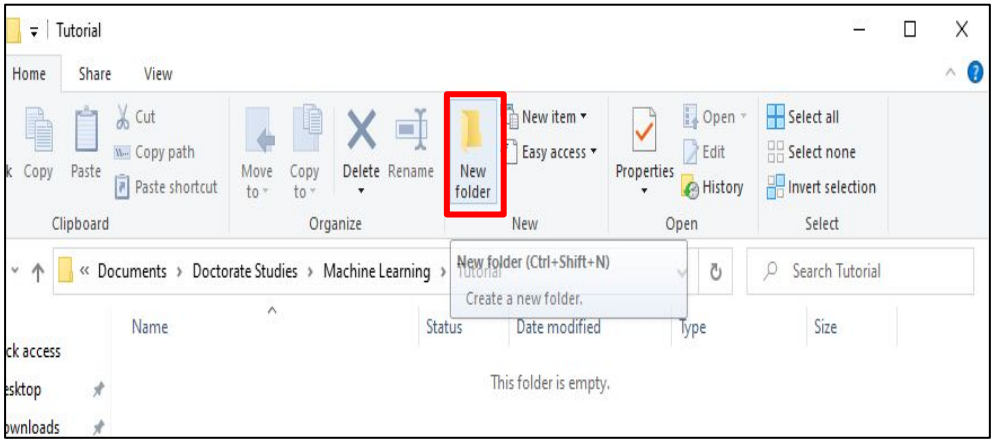
mkdir *<directory>*

Example:

```
Ajay9393@sargent_server:~$ ls
auto_ML  examples.desktop  nohup.out  test.py
Ajay9393@sargent_server:~$ mkdir ML_Tutorial
Ajay9393@sargent_server:~$ ls
auto_ML  examples.desktop  ML_Tutorial  nohup.out  test.py
Ajay9393@sargent_server:~$
```

Created a directory called ML_Tutorial in the current directory (still *home/Ajay9393*)

In Windows, this is equivalent to clicking the “new folder” icon, and then renaming the folder to *<directory>*



Moving around: the *cd* command

Linux command

File or Directory being operated on by command

Syntax:

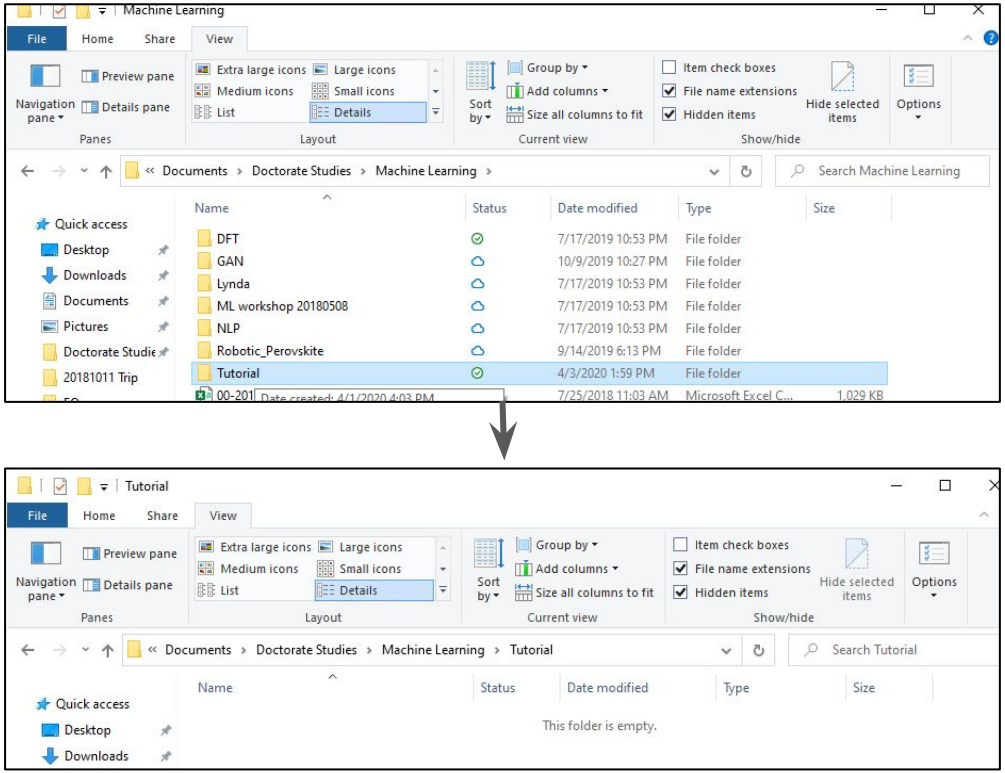
cd *<directory>*

Example:

```
Ajay9393@sargent_server:~$ cd ML_Tutorial/
Ajay9393@sargent_server:~/ML_Tutorial$ pwd
/home/Ajay9393/ML_Tutorial
Ajay9393@sargent_server:~/ML_Tutorial$
```

Move into the ML_Tutorial folder, and then print working directory (pwd) to see that we've moved!

In Windows, this is equivalent to "double clicking" a folder



Some special *cd* commands

Linux command

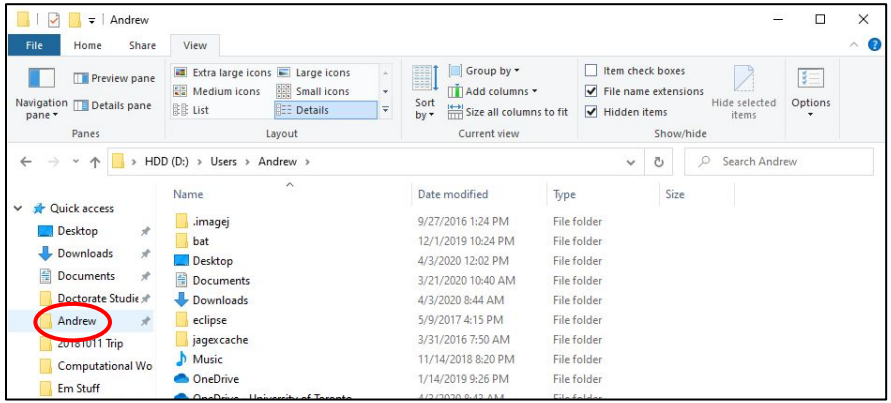
File or Directory being operated on by command

1. **Move to home directory**

   Syntax:

   cd ~

```
Ajay9393@sargent_server:~/ML_Tutorial$ pwd
/home/Ajay9393/ML_Tutorial
Ajay9393@sargent_server:~/ML_Tutorial$ cd ~
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$
```

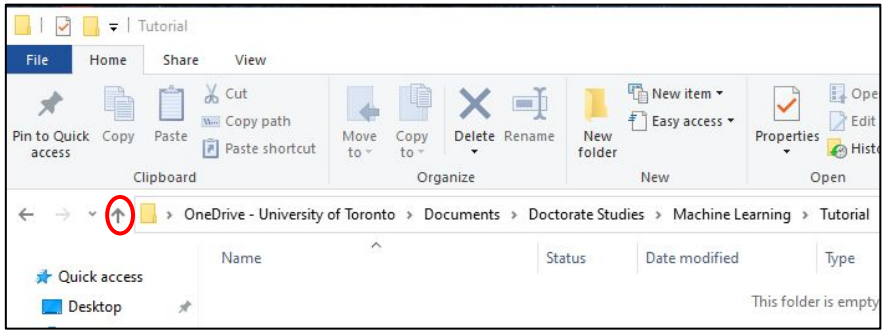


2. **Move to parent directory:**

   Syntax:

   cd ..

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ cd ..
Ajay9393@sargent_server:/home$ pwd
/home
Ajay9393@sargent_server:/home$
```



Moving multiple directories at once

- First, let's create a few new directories using mkdir and cd:
  - Move to the ML_Tutorial folder
  - Create a new folder named "New_Folder1"
  - Move into the "New_Folder1"
  - Create a new folder called "New_Folder2"
  - Move back to the *root* directory

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ cd ML_Tutorial/
Ajay9393@sargent_server:~/ML_Tutorial$ pwd
/home/Ajay9393/ML_Tutorial
Ajay9393@sargent_server:~/ML_Tutorial$ mkdir New_Folder1
Ajay9393@sargent_server:~/ML_Tutorial$ ls
New_Folder1
Ajay9393@sargent_server:~/ML_Tutorial$ cd New_Folder1/
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1$ mkdir New_Folder2
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1$ ls
New_Folder2
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1$ cd ~
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$
```

Moving multiple directories at once

- If you need to navigate a complex file system, it becomes tedious to repeatedly type in *cd <next_folder>*
- Fortunately, you can accomplish this all in one line!
- Hint: Use the TAB button→ it is autocomplete and is a lifesaver

Syntax:

➢ ***cd*** *<directory1/directory2/directory3…./directoryN>*

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ cd ML_Tutorial/
Ajay9393@sargent_server:~/ML_Tutorial$ cd New_Folder1/
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1$ cd New_Folder2/
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1/New_Folder2$ pwd
/home/Ajay9393/ML_Tutorial/New_Folder1/New_Folder2
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1/New_Folder2$
```

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ cd ML_Tutorial/New_Folder1/New_Folder2/
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1/New_Folder2$ pwd
/home/Ajay9393/ML_Tutorial/New_Folder1/New_Folder2
Ajay9393@sargent_server:~/ML_Tutorial/New_Folder1/New_Folder2$
```

directory1/directory2/directory3…./directoryN is what is known as a ***path***

Paths

- A path specifies the location of *a file or a directory*
- There are two types of paths: *absolute* and *relative*
- An absolute path specifies the *absolute* location of a file or directory, and can be called from wherever. These will begin with a **forward slash (*/*)**
- A relative path specifies the location of a file or directory *relative* to the current directory. These will **not begin** with a forward slash

The power of paths

Any file or directory can be referenced by a linux command via its *absolute* or *relative* path.
Consider the following:

```
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ ls ML_Tutorial/New_Folder1/
New_Folder2
Ajay9393@sargent_server:~$ mkdir ML_Tutorial/New_Folder1/New_Folder2b
Ajay9393@sargent_server:~$ ls ML_Tutorial/New_Folder1/
New_Folder2  New_Folder2b
Ajay9393@sargent_server:~$
```

For the rest of the tutorial, files and directories may be referenced by their path if it makes our lives easier!

1. Check which directory we are in with ***pwd***
2. List the files and folders with ***ls*** in the *Ml_Tutorial/New_Folder1/* directory
   a. *New_Folder2*
3. Create a new folder with ***mkdir*** called *New_Folder2b* in the *Ml_Tutorial/New_Folder1/*
4. List the files and folders with ***ls*** in the *ML_Tutorial/New_Folder1/* directory
   a. *New_Folder2*
   b. *New_Folder2b*

Summary so far

- Linux is like Windows or macOS, but uses a text-based interface
- You can log into a Linux server with a **S**ecure **Sh**ell connection
- There are several basic ***commands*** for navigating linux directories:
  - ***pwd***
  - ***ls***
  - ***mkdir***
  - ***cd***
- These commands operate on ***files or directories***, which may be referenced by their *absolute* or *relative* **path**
  - Absolute paths begin with a forward slash (/)
  - Relative paths do not begin with a forward slash (/)
- Commands may be modified with ***options***, which are passed when the ***command*** is called by using ***-option***

Creating and Editing text files - introducing vim!

Vim is a text editor that is pre-installed on most linux systems (Note: it is an improved version of VI, which was the original Linux text-editor)

Vim is going to be incredibly frustrating to use if you are used to Windows or Mac. If you want to be proficient **practice!**

The biggest difference is that you can't click with your mouse to move the cursor. You need to use your keyboard to move the cursor to insert text at different places.

Creating a new text file

Linux command

File or Directory being operated on by command

Syntax:

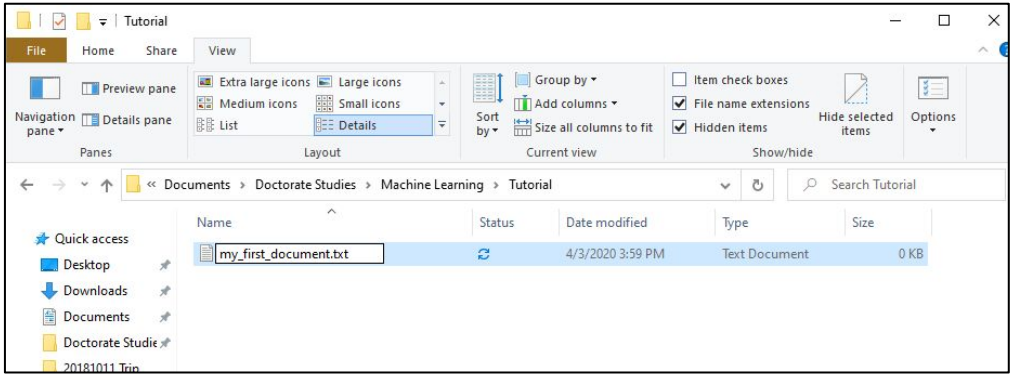
vim <filename>

Example:

```
Ajay9393@sargent_server:~$
Ajay9393@sargent_server:~$ cd ~
Ajay9393@sargent_server:~$ pwd
/home/Ajay9393
Ajay9393@sargent_server:~$ vim /home/Ajay9393/ML_Tutorial/my_first_document.txt
```

This command will open a text document called my_first_document.txt in the ML_Tutorial folder when I hit enter. If it does not exist, it will create it. Note: Absolute path used to create this file

In Windows, this is like if we created a new text document, and then opened it in notepad.



Important Commands in vim

To enter a command in vim, you press a letter

- ***i*** will enter the *insert mode*
- ***v*** will enter *visual mode*

Hit “esc” to exit a particular mode

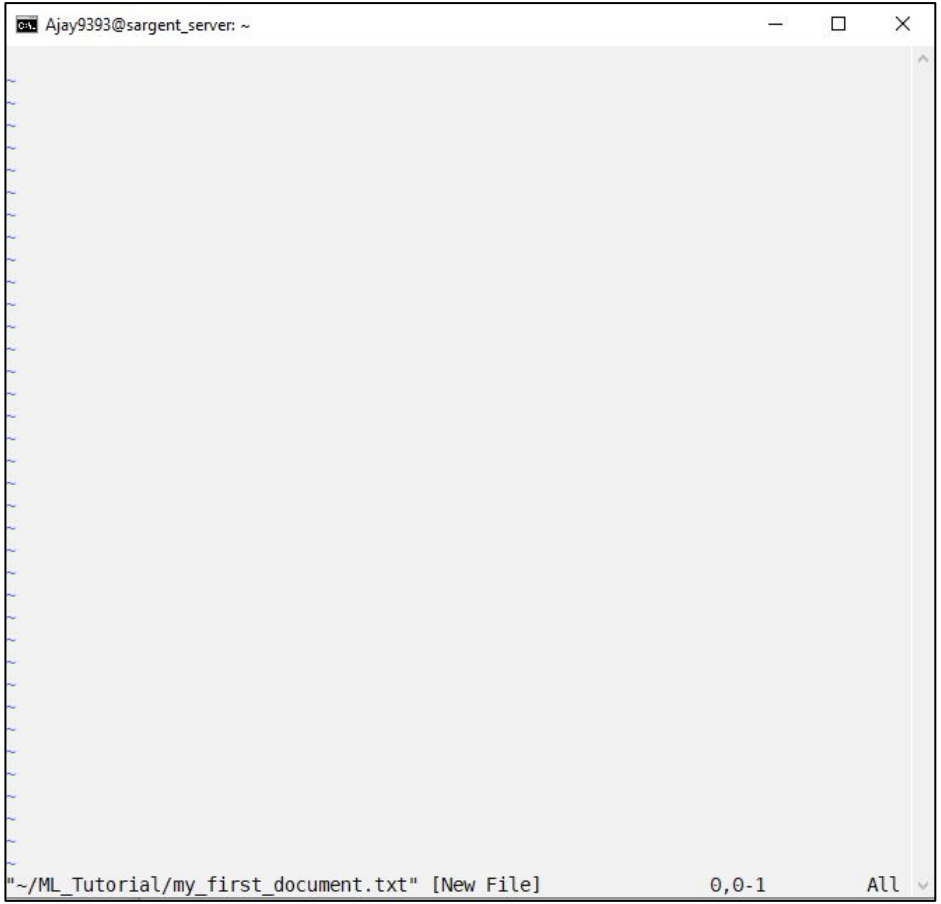
Importantly, to close open documents in VIM:

- ***:q + Enter*** will quit out of a document that has not been changed
- ***:wq + Enter*** will save changes to the document and quit
- ***:q! + Enter*** will quit the document and overwrite

Walkthrough of opening and typing in vim

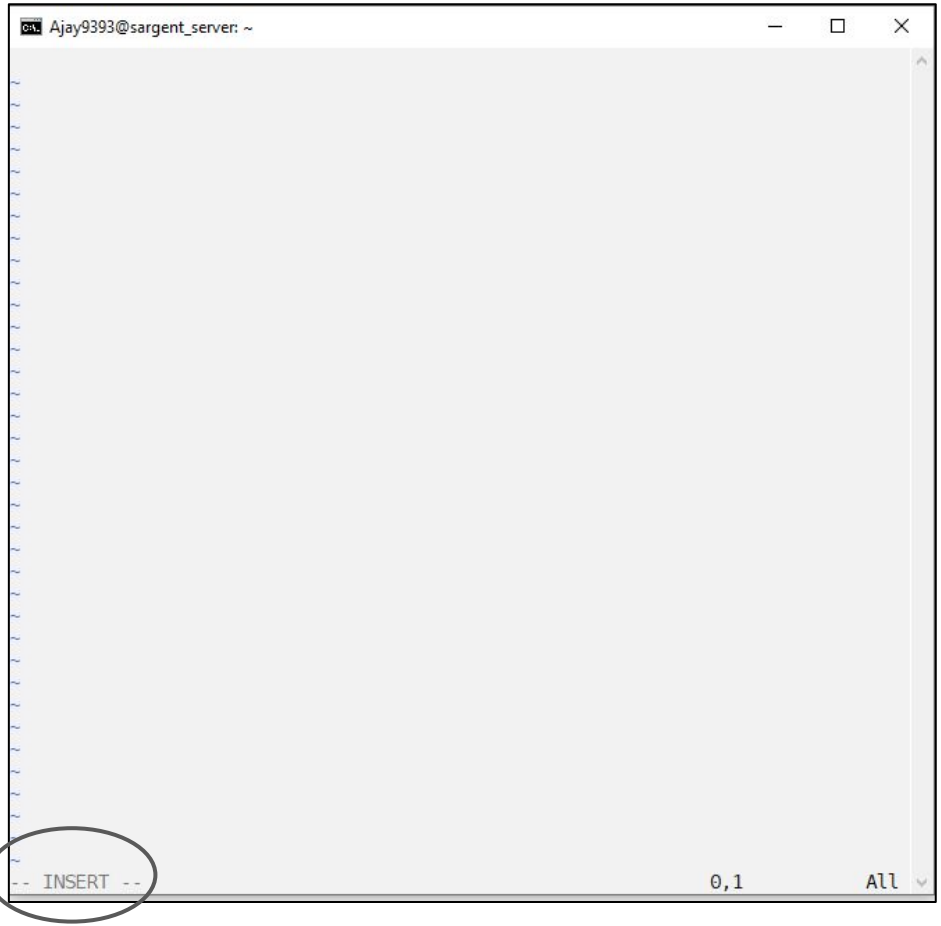
There are many MANY commands in vim that all serve different functions.
The most important thing you will do in vim is to enter *insert mode (i)* and add text.
And then, if you like what you added, save the document and close it (*:wq)*. If you don’t like what you added, exit and DON’T save *(:q!)*.

A new document



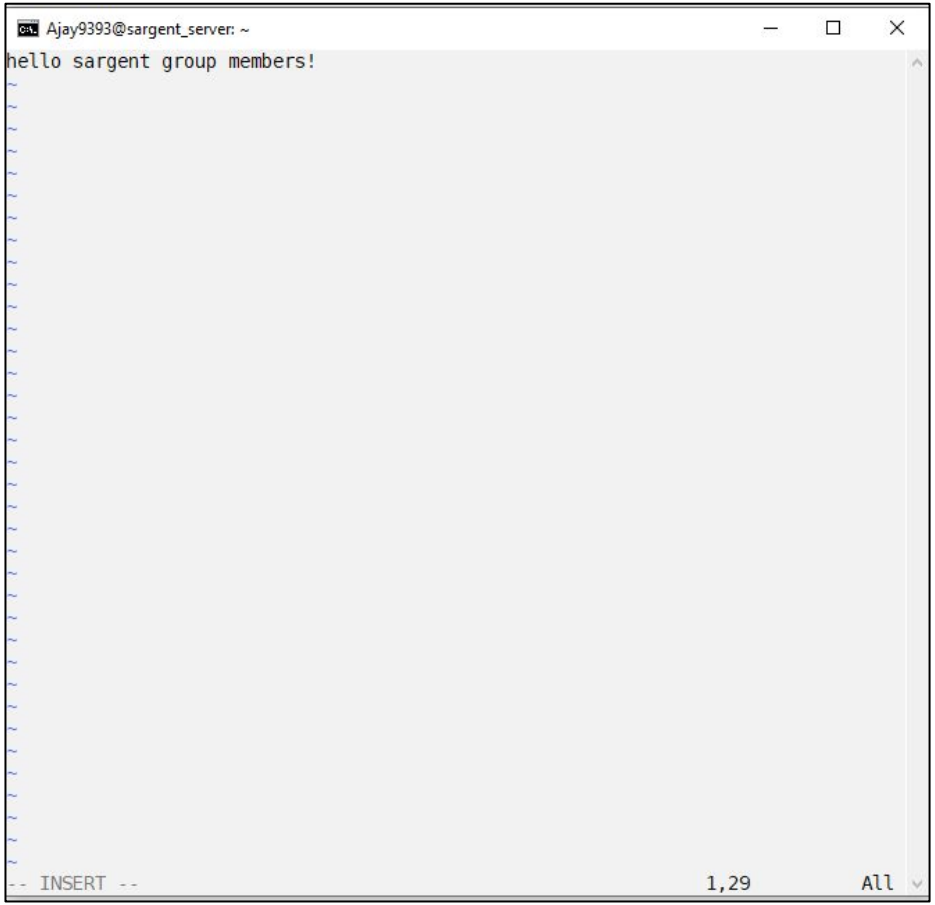
At this point, I am in the read-only mode of vim. I cannot add any text until i hit “i”.

Insert Mode



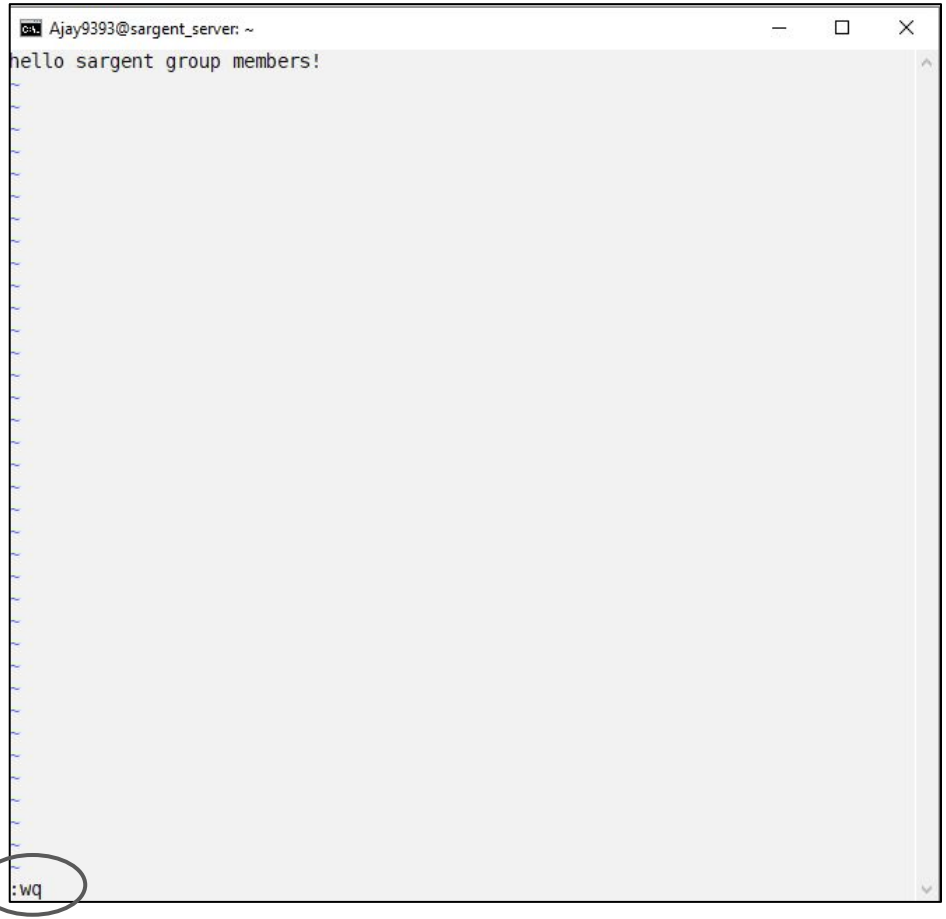
Now i can add text, because i am in the “insert mode”

Typing



After i am done typing, i need to hit “esc” to exit the insert mode. As I want to save the document, I will then type “***:wq***” to exit

Saving and closing



You can see that i am no longer in insert mode - now the command that i am typing into vim appears in the bottom of terminal window, where it previously said “insert”. After i hit enter, the document will be saved!

If i now check to see that the document is there, it is!

```
Ajay9393@sargent_server:~$ ls /home/Ajay9393/ML_Tutorial/
my_first_document.txt  New_Folder1
Ajay9393@sargent_server:~$
```

Let’s make a python file

- Create a .py file using “vim test.py”
- Let’s make it count from 1 to 10:
  - Enter insert mode
  - Type:
    - for i in range(1,11):
  - Go to the next line
  - Add four spaces (or a TAB)
  - Type:
    - print (“I am counting to 10! I am at {}”.format(i))
- Save the file:
  - Esc
  - :wq + enter
- And try running it!
  - Type:
    - python test.py
    - Hit enter!

Other useful Vim commands

- v: visual mode. In this mode, as you move the cursor you will highlight text.
- y/(shift+y): copy the (highlighted/entire line) text
- dd: cut the entire line
- p/(shift+p): paste (after/before) the cursor position
- /[TEXT]: search for text matching this string. To go to the next instance of the text, hit “n”
- Ctrl+ f: scroll down one page
- Ctrl+b: scroll up one page
- gg: go to top of document
- Shift+g: go to bottom of document
- [NUM]gg+enter: go to line NUM

Let’s Practice!

Open the “data_and_calcs.py” file that is in your home directory.

And now let’s navigate around the text document!

Vim tips and tricks

- We aren’t going to spend any more time on Vim - it needs to be learned in small doses
- Copying and pasting from windows clipboard into vim is tricky, and should be avoided when possible
- You can use this cheat sheet for all of the basic commands: https://vim.rtorr.com/
- It is tough! Don’t give up
- Unless you are developing code on the supercomputing clusters (i.e ML for computationalists), you are likely going to be okay with the very easy vim commands. Most DFT calculations can be set up manually with fairly straight forward text files

Deleting files: the *rm* command

Linux command

File or Directory being operated on by command

Options to Linux Command

The command “rm” stands for remove. It can be used to delete files or folders from the servers.

Syntax:

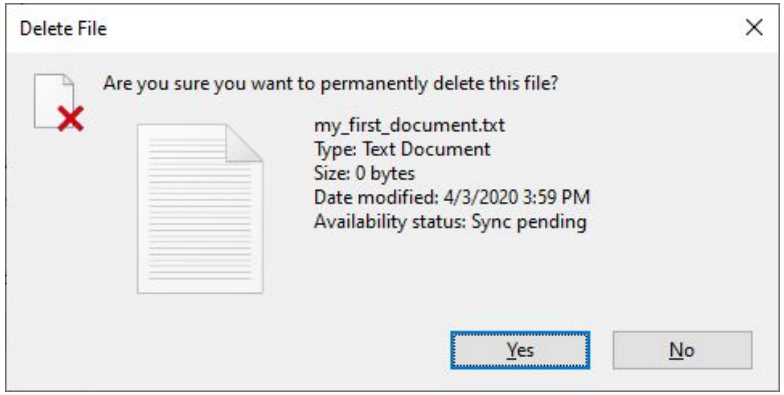
rm [options] <directory>

Example:

```
Ajay9393@sargent_server:~$ ls ML_Tutorial/New_Folder1/
New_Folder2  New_Folder2b
Ajay9393@sargent_server:~$ rm -r ML_Tutorial/New_Folder1/New_Folder2
Ajay9393@sargent_server:~$ ls ML_Tutorial/New_Folder1/
New_Folder2b
Ajay9393@sargent_server:~$
```

Note: the “***-r***” option means recursive, and **must be used if deleting a directory**, but is not needed for files

In Windows, this is equivalent to...deleting a folder or file



Moving and Copying files: *mv* and *cp*

Linux command

File or Directory being operated on by command

Options to Linux Command

The commands "*mv*" and "*cp*" stand for move and copy, respectively. As the syntax is the same, we address them together

Syntax:

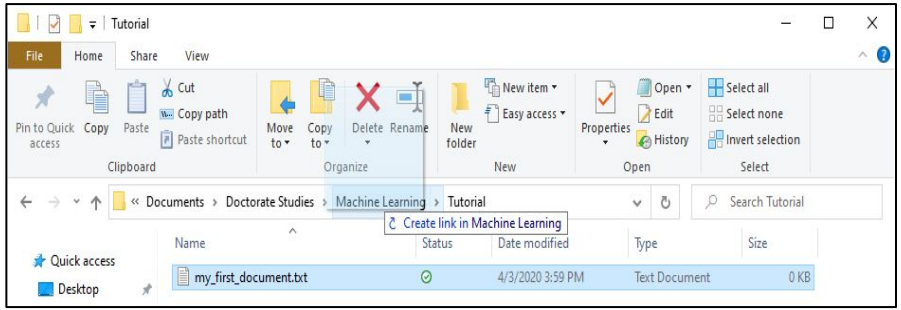
***mv* <directory1> <directory2>**

Example:

```
Ajay9393@sargent_server:~/ML_Tutorial$ pwd
/home/Ajay9393/ML_Tutorial
Ajay9393@sargent_server:~/ML_Tutorial$ ls
my_first_document.txt  New_Folder1
Ajay9393@sargent_server:~/ML_Tutorial$ mv my_first_document.txt New_Folder1/
Ajay9393@sargent_server:~/ML_Tutorial$ ls
New_Folder1
Ajay9393@sargent_server:~/ML_Tutorial$ ls New_Folder1/
my_first_document.txt  New_Folder2b
Ajay9393@sargent_server:~/ML_Tutorial$
```

Note: the "***-r***" option means recursive, and **must be used if deleting a directory**, but is not needed for files

In Windows, this is equivalent to dragging a file from one folder into another, or copying and pasting it!



Practice

We have uploaded a practice document for you to mess around with - we can make solutions for them as well if you would like, but we believe you will be able to figure it out :)

Again, here is a great, more in-depth tutorial: https://ryanstutorials.net/linuxtutorial/

Other Questions?

Python

Overview

- What is python?
- How to install python
  - Linux
  - Windows
  - Mac
- Conda Environments
- Jupyter notebooks
- Google Collab notebooks
- Your first function!
- Useful libraries for ML
  - Numpy
  - Pandas
  - Sklearn

Schedule Today

- This lecture is longer than the previous one
- We will take a break at ~10:45 for 15 minutes, and another one at ~12:15 for 20 minutes.
- We will leave some of the lecture for tomorrow, particularly the parts pertaining to using python libraries.

What is Python?

Python is an extremely common **programming language.**

This means that python allows users to write a set of commands, and the *python interpreter* will figure out what they mean and execute those commands.

As far as programming goes, python is easy to read and understand because the *syntax* is fairly straightforward.

Syntax is a fancy way of saying language - it just means that python code has to be written following certain rules, which we will talk about!

How can python be useful to you?

- Python is a rapidly growing language - and this is primarily because of Machine Learning!
- Popular machine learning libraries, such as Tensorflow, are written for python.
- Fortunately, most of this is easy to use - with a little bit of practice, you can apply ML to your own experiments!

What you should take from this course

- Understanding basic python scripts
  - Syntax
  - Manipulation of python objects
  - Using loops
  - Writing functions
- Being able to import and use standard python libraries
- Using Google Colab notebooks

Python Setup

- Generally, you have to install python before it can be run
- This can be done from python.org
- However, it is better to download python from Anaconda:https://www.anaconda.com/
- Anaconda is a *package manager*, which means it will automatically manage different *library compatibilities*
  - Don’t worry about this right now, just trust me that it is better to install python with Anaconda!

Installing python with Anaconda - Windows

From: https://repo.continuum.io/archive/, find most recent download

Anaconda3-2020.02-Windows-x86.exe	423.2M	2020-03-11 10:32:58	64ae8d0e5095b9a878d4522db4ce751e
Anaconda3-2020.02-Windows-x86_64.exe	466.3M	2020-03-11 10:32:35	6b02c1c91049d29fc65be68f2443079a

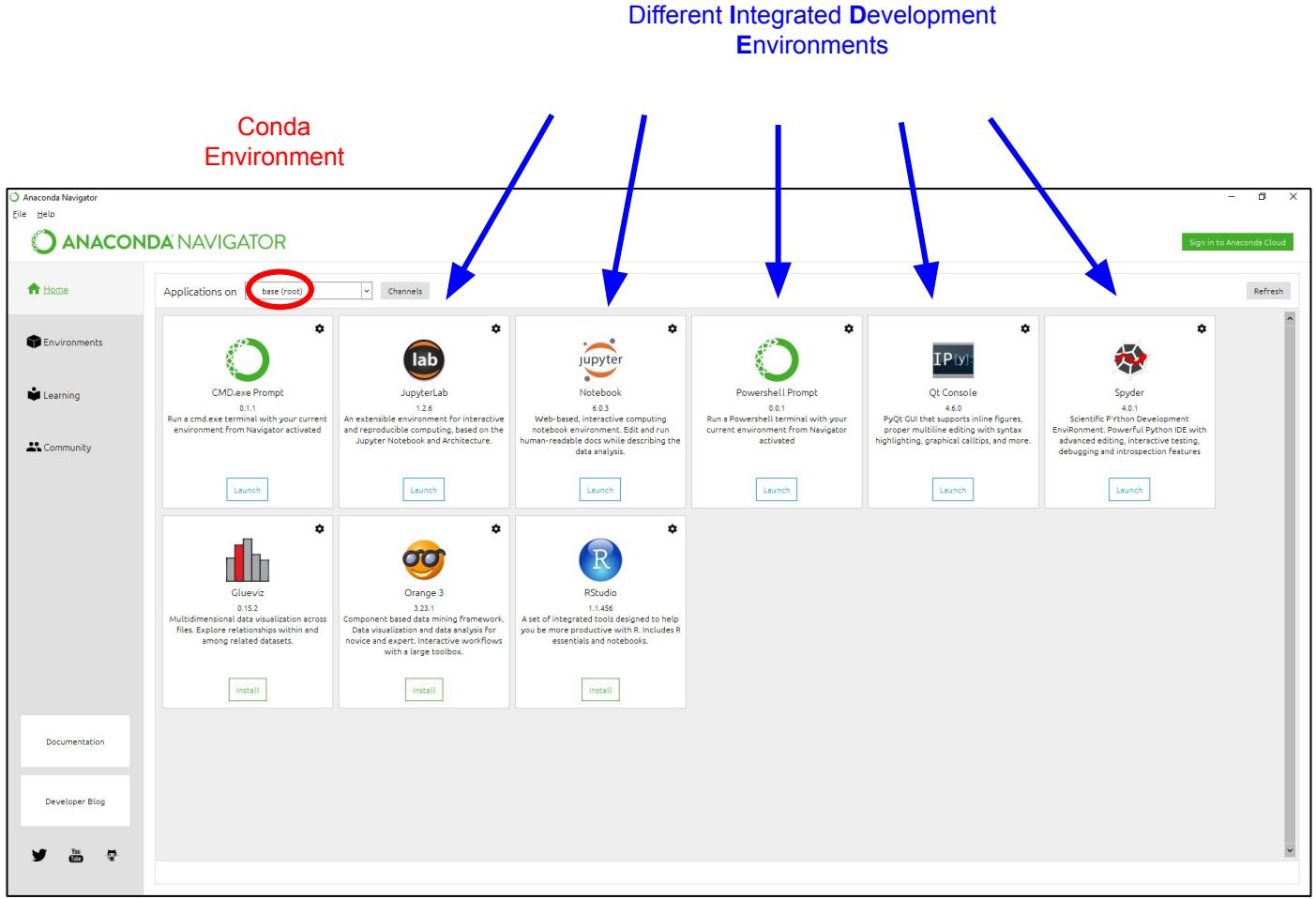
Click the link and run the executable. Follow the prompts (default installation).

The Anaconda Navigator

There will be a default menu that looks like this.

You can download and run **any** of the integrated development environments, such as Spyder, (fancy text editors) to practice coding!

This is kind of like Matlab, or Excel. You run a program, and then do things in that program

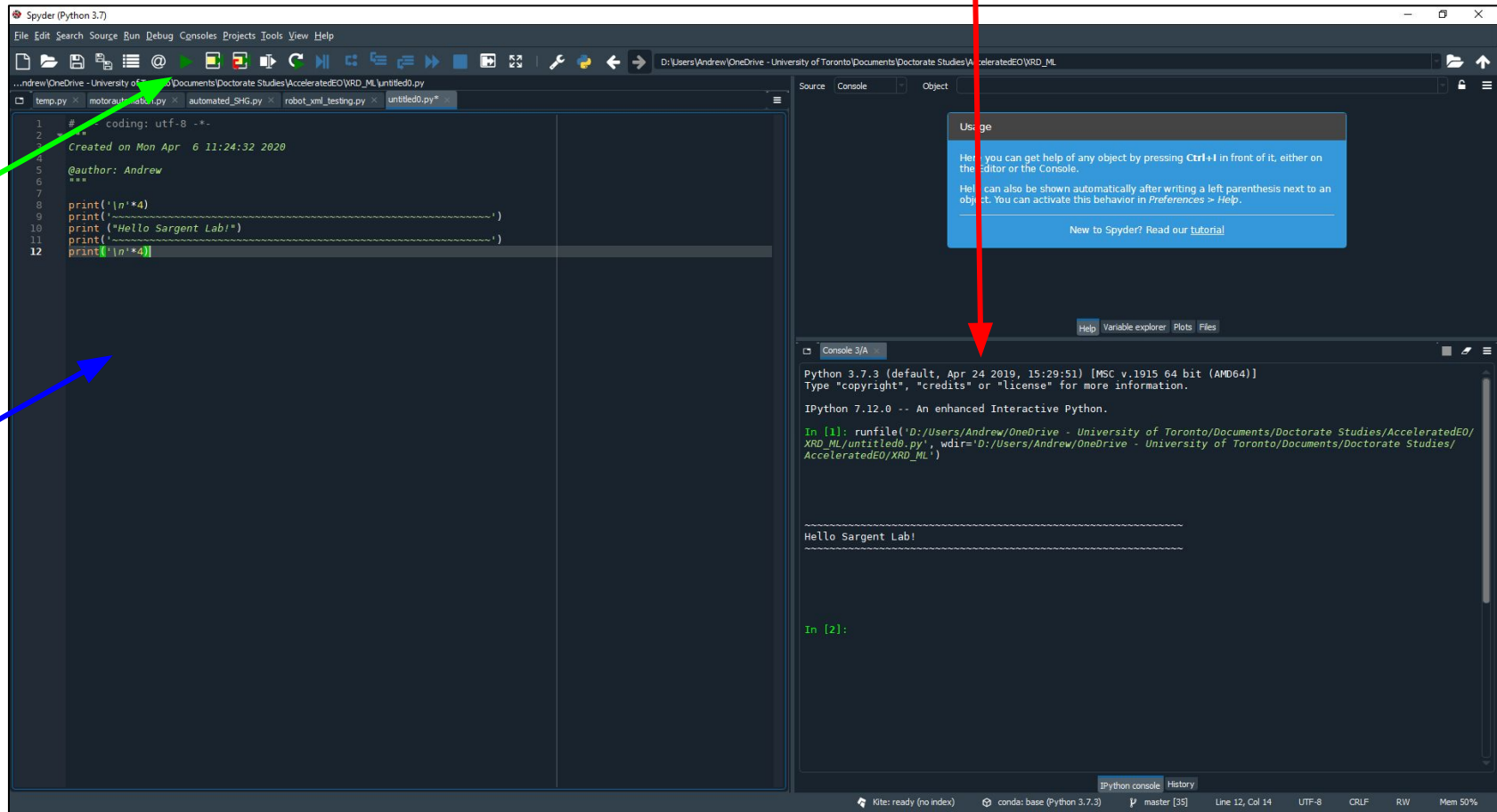


Example of an IDE: Spyder

Output console (and input for debugging!)

Run code in text editor

Text Editor



Installing Anaconda - Mac & Linux

- Two options for installation:
  a. Follow the instructions here: https://docs.anaconda.com/anaconda/install/mac-os/
  b. From the link (https://repo.continuum.io/archive/), download the latest version of Anaconda

Anaconda3-2020.02-MacOSX-x86_64.pkg	442.2M	2020-03-11 10:32:57	d1e7fe5d52e5b3ccb38d9af262688e89
Anaconda3-2020.02-MacOSX-x86_64.sh	430.1M	2020-03-11 10:32:34	f0229959e0bd45dee0c14b20e58ad916

- For linux distributions, we recommend the user to follow following steps:
  - Download the appropriate version from (https://repo.continuum.io/archive/); for the time being, call it script.sh
  - Change the permissions of the executable to execute by: chmod u+x script.sh
  - Execute the script by the command: ‘./script.sh’. This will begin the installation process.
  - Follow the steps numbered 4 (included) & onwards as listed out on this link: https://docs.anaconda.com/anaconda/install/linux/

Conda Environments

- A great resource because they allow you to have different versions of pythons and different versions of python libraries
- Conda will take care of all package dependencies and other such nonsense
- You will only need to use these IF you want to do heavy python development on a cluster
- While powerful, Conda is not super user-friendly, and it is not feasible to instruct everyone in its use in a single session. All you need to know is here: https://docs.conda.io/projects/conda/en/latest/user-guide/
- Ask Andy or Hitarth for help getting started if you want to install python locally and begin using it all the time!

Making all of the above redundant: Google Colab

- Google colab is a free cloud-based ipython notebook.
- This means that it can be used to write and execute python code, and the computations will be carried out using cloud computing:
  - You are not limited by the awful computer Remi gave you!
  - It uses GPU computing which means that it is ideal for complex ML networks
- You can perform a calculation for up to 12 h continuously
- You have (more-or-less) unlimited access to it
- It can be done offline as well

Getting started with Google Colab

You should all have created a blank notebook (as per instructions before course started)

These are in a Google Drive account→ telling python to access code from these Collab notebooks will be the same as if you were on Windows (we'll get to this).

Similarly to yesterday, we will be going through the notebook step-by-step, and encourage each of you to follow along!
A great python reference: https://www.tutorialspoint.com/python/index.htm

Setting up Google Colab Notebooks!

Notebooks

- User-friendly coding interface
- “Write and test” philosophy
- It was first developed as part of **Project Jupyter** (https://jupyter.org/)

```
[ ]  # The downloaded file is a csv file.
     # We will use Pandas' read_csv module to read the file
     import pandas as pd
     df = pd.read_csv('12978425')
     print(df.head())
```

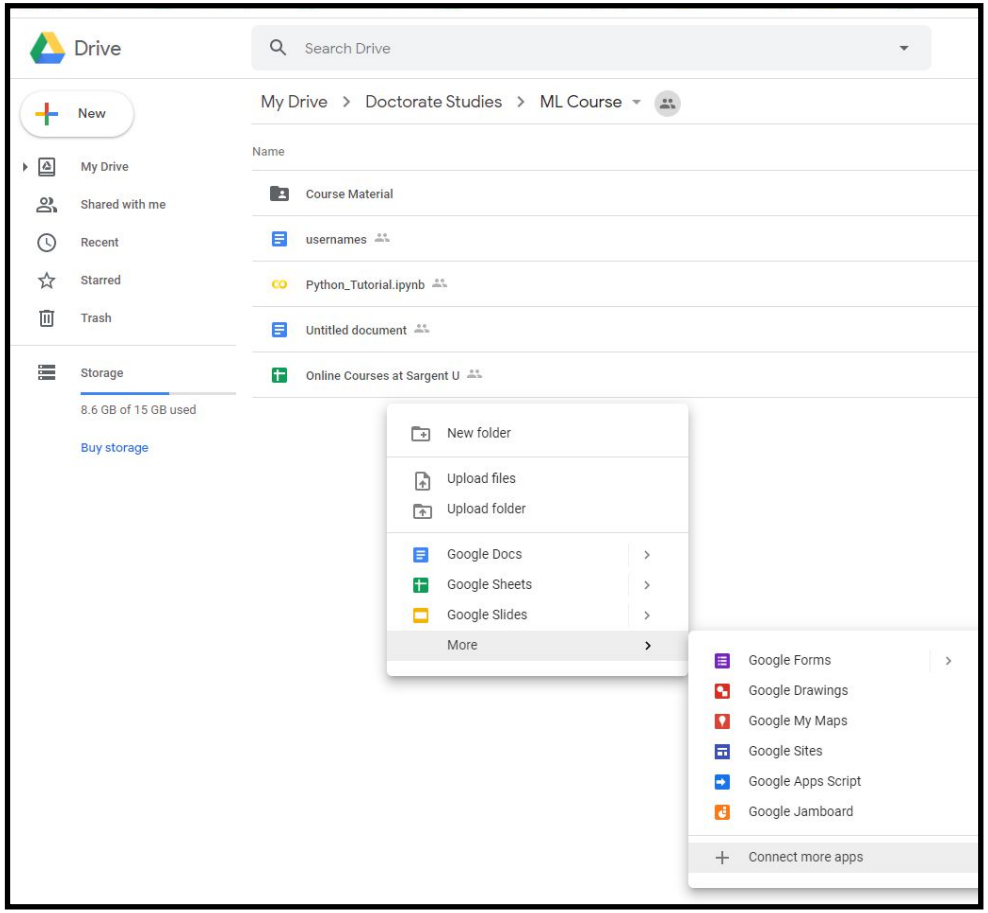
```
  Material Composition A site #1  ... Asite_IsAlkali_max Bsite_IsMetal_max
0          Ba1Sr7V8O24        Ba  ...                  1                 1
1      Ba2Bi2Pr4Co8O24        Ba  ...                  1                 1
2         Ba2Ca6Fe8O24        Ba  ...                  1                 1
3      Ba2Cd2Pr4Ni8O24        Ba  ...                  1                 1
4         Ba2Dy6Fe8O24        Ba  ...                  1                 1

[5 rows x 81 columns]
```

We can look at the columns of the dataframe with 'df.columns.values'

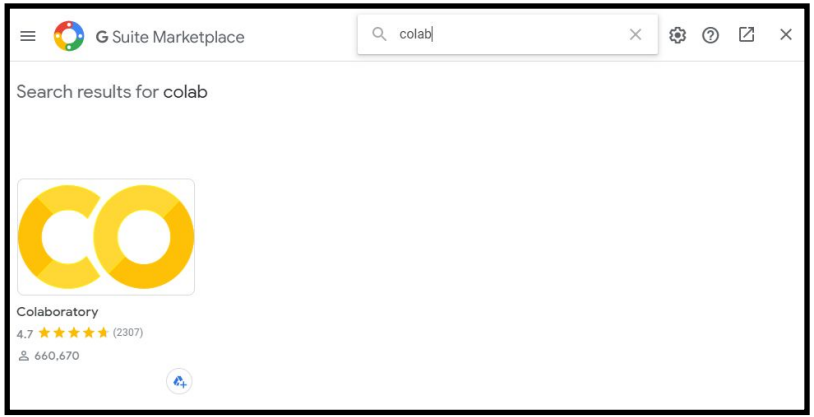
```
[ ]  df.columns.values
```

```
array(['Material Composition', 'A site #1', 'A site #2', 'A site #3',
       'B site #1', 'B site #2', 'B site #3', 'X site',
```

Install Google Colab



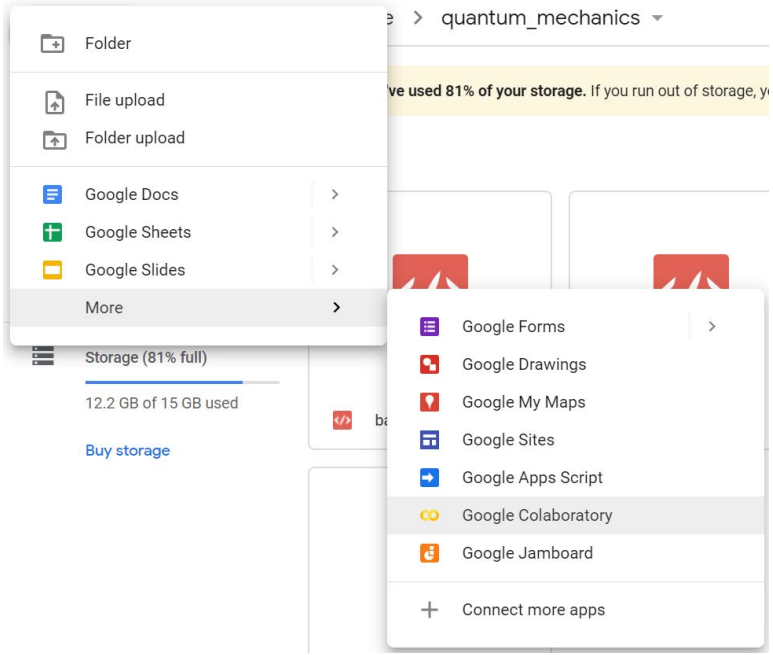
- In a folder, go to “New” → “More” → “Connect more apps”
- Search “colab” and install it:



Google Colab

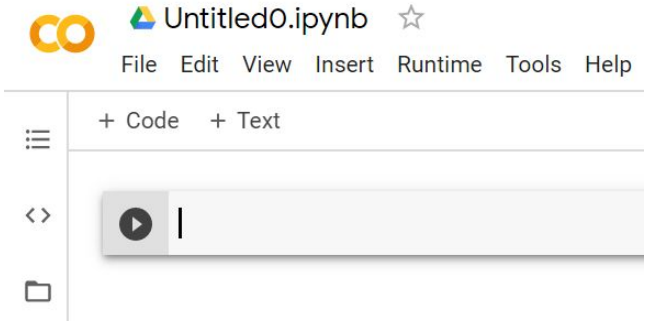
- Go to your google drive (https://drive.google.com/)
- Make a new folder - name it
  - New -> Folder
  - ‘Quantum_mechanics’ for example
  - Try to avoid any spaces in your names
  - Ex: ‘Hitu_24’ is highly preferred over ‘Hitu 24’
  - Makes it very difficult to navigate cross platforms

- Select new -> more -> Google Colaboratory

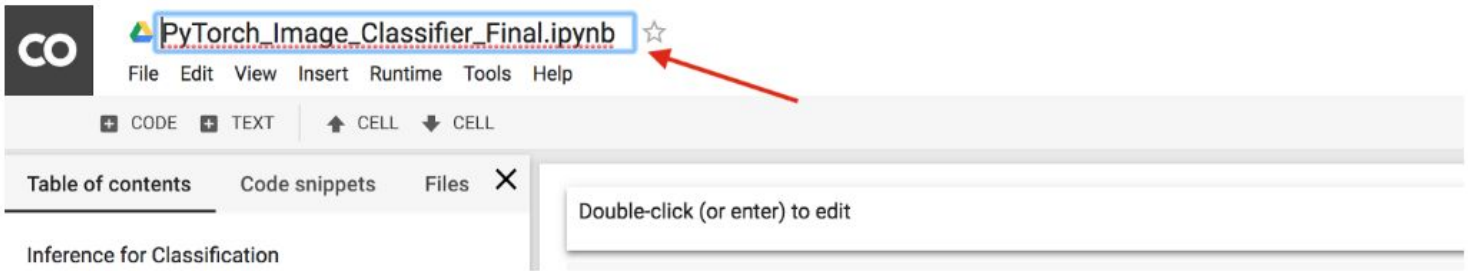


Your First Notebook

- You will see the following window pop up as new tab in your browser:

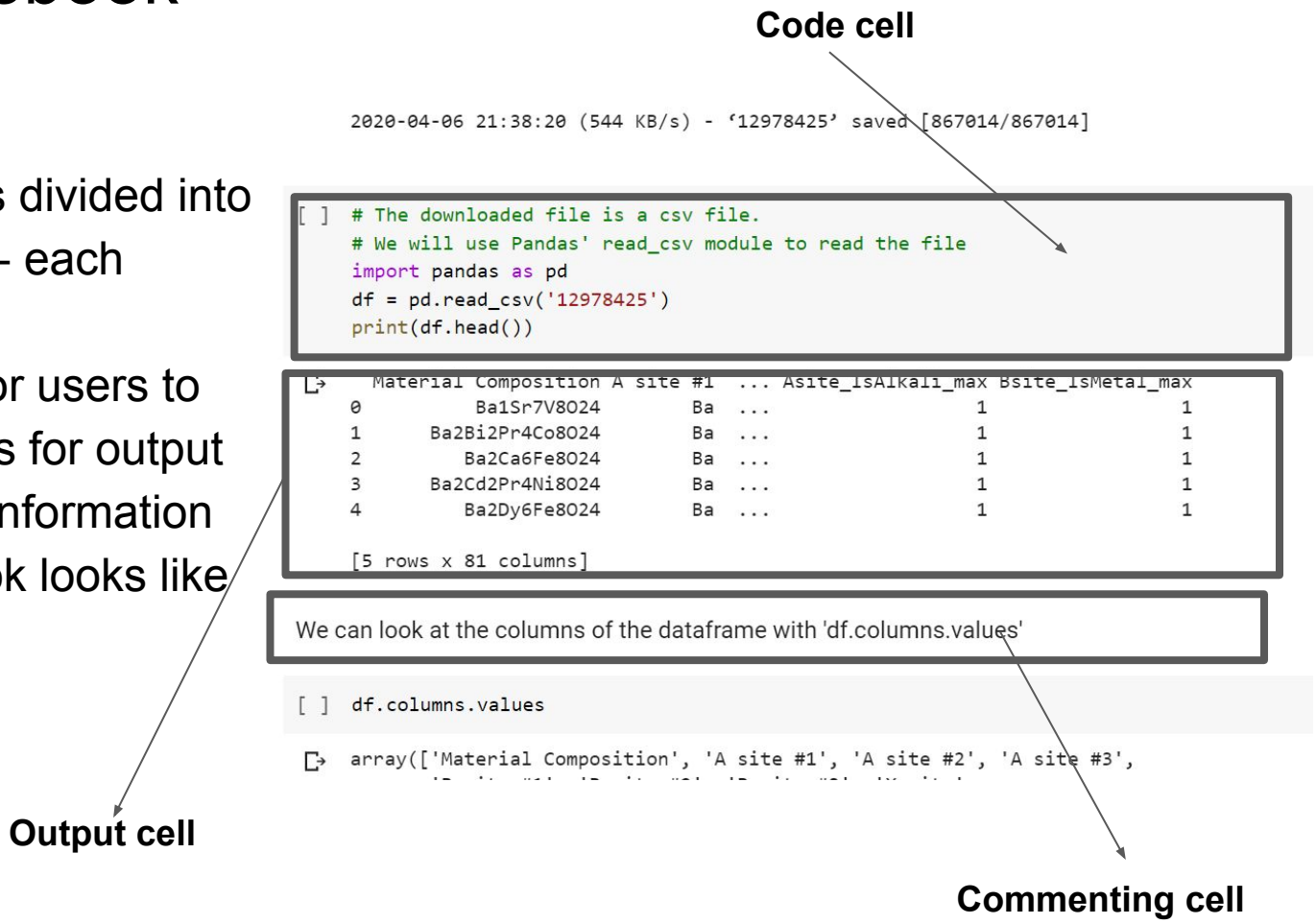


- This is your new notebook
- You can rename your notebook to whatever name you want (again try to avoid any spaces in your file / notebooks / folder names)



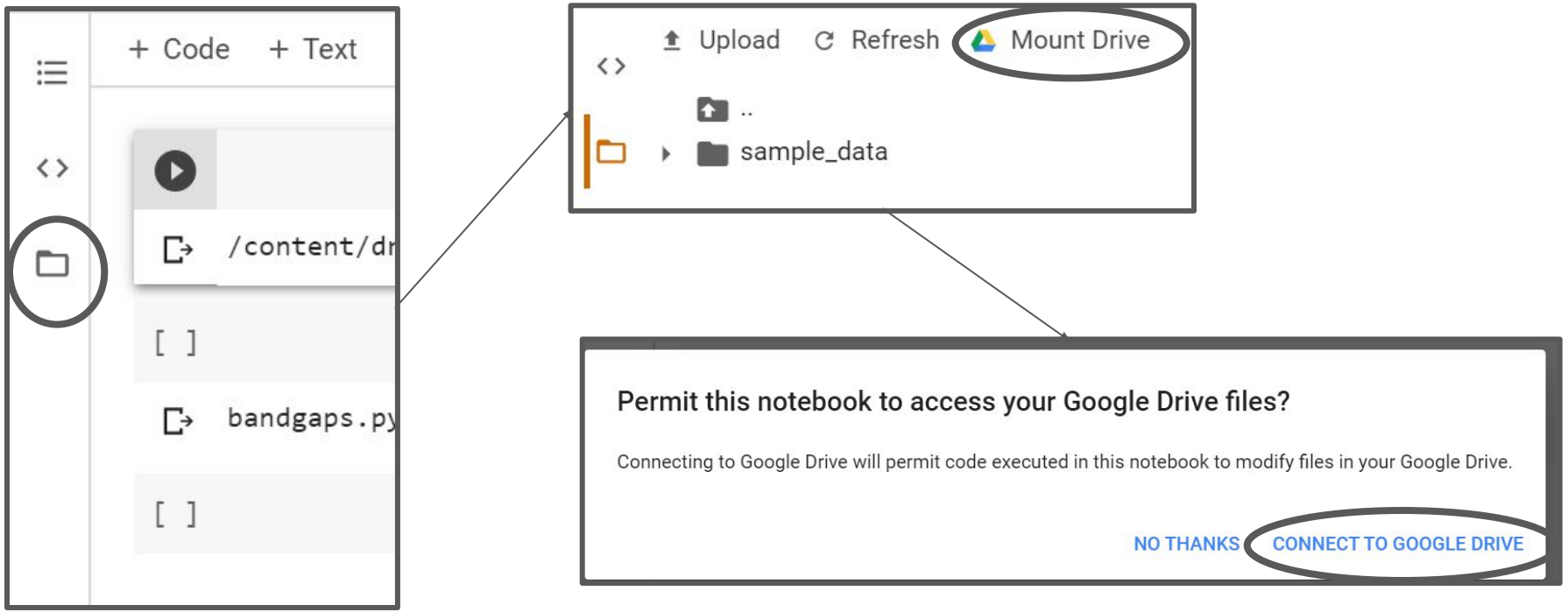
Layout of a Notebook

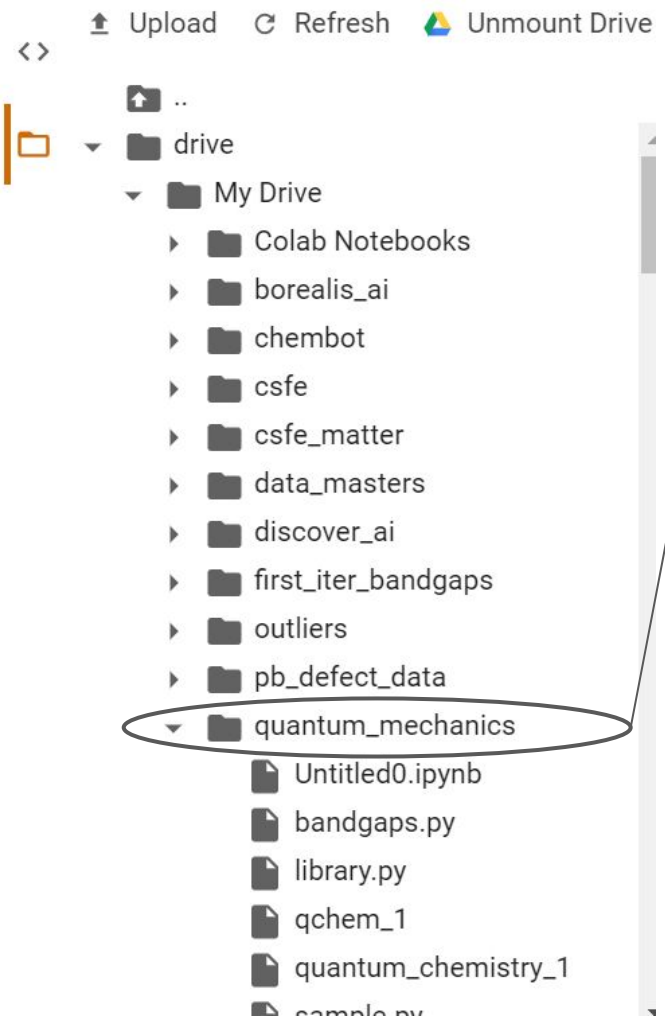
- Each notebook is divided into several sections - each called a **cell**
- Some cells are for users to write code; others for output or comments or information
- A typical notebook looks like this:



Mounting GDrive

- Can you access your files present in your Google Drive? Yess!





The directory you created can be located as: drive -> My Drive -> quantum_mechanics

To access the files stored in your folder, we will use ‘cd’ command of linux to change to the required folder as shown:

```
[2]  cd drive/My\ Drive/quantum_mechanics

     /content/drive/My Drive/quantum_mechanics

[3]  pwd

     '/content/drive/My Drive/quantum_mechanics'
```

‘pwd’ command gives us our current location.

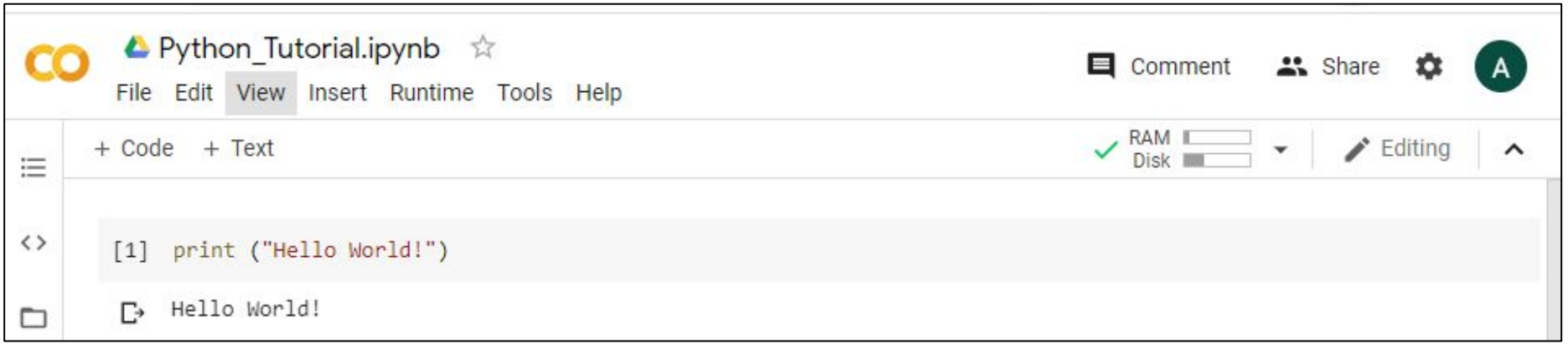
As you can see, you are in the required folder! You will have access to the all the files here without stating any relative paths now!

Now you are all set to run your first Colab Notebook!

The most common coding phrase in the world

Let’s start by saying hello!

Type: print (“Hello World!”) and run the cell (shift+Enter)



One of the most common - and useful - functions in Python is the print() command. This is used to update, debug, and so much more!

Some notes on Python Syntax

- Code is executed *line-by-line*
- # indicates a comment (NOT executed by Python)
- Python uses *indents* to group bits of code together (example on next page), which means that *whitespace* is important
- Functions in python are called with the following syntax :

  Output = **Function**(*Input*)

  And can be thought of as just manipulating an input. Functions do not necessarily require either an input OR an output. In the case where there is no output (i.e, the *print* function), the function can just be called. If there is no *input,* the function is called with closed brackets, i.e:

  Output = **Function**()

Example of Indenting in Python

These lines indicate which “block” of code a line belongs to. In google collab, there is an option to have these lines displayed or not

Indentation is used to replace the “end” function in Matlab to indicate when a specific block of code is finished

```
[13] ##demonstrating how indents work
     ##don't worry about the logic of what is happening below, just note how different "blocks" are grouped
     for i in range(1,11):
         ##all code that is indented once is part of the for loop
         print(i)
         if i >5:
             ##all code indented twice is part of the "if" statement and the for loop
             print('We are more than halfway there!')
         #Back to one indent--> exited the "if" block, and entering an "elif" block
         elif i < 5:
             ##we are now in the "elif" code block
             print ("We are not halfway there yet!")
         #exited the elif block
         else:
             ##Now we exited the "elif" and entered the "else" code block
             print ("We are exactly halfway there!")

     ##When a line of code has no indents, it means we have now left the for loop block
     print('All Done!')
```

```
1
We are not halfway there yet!
2
We are not halfway there yet!
3
We are not halfway there yet!
4
We are not halfway there yet!
5
We are exactly halfway there!
6
We are more than halfway there!
7
We are more than halfway there!
8
We are more than halfway there!
9
We are more than halfway there!
10
We are more than halfway there!
All Done!
```

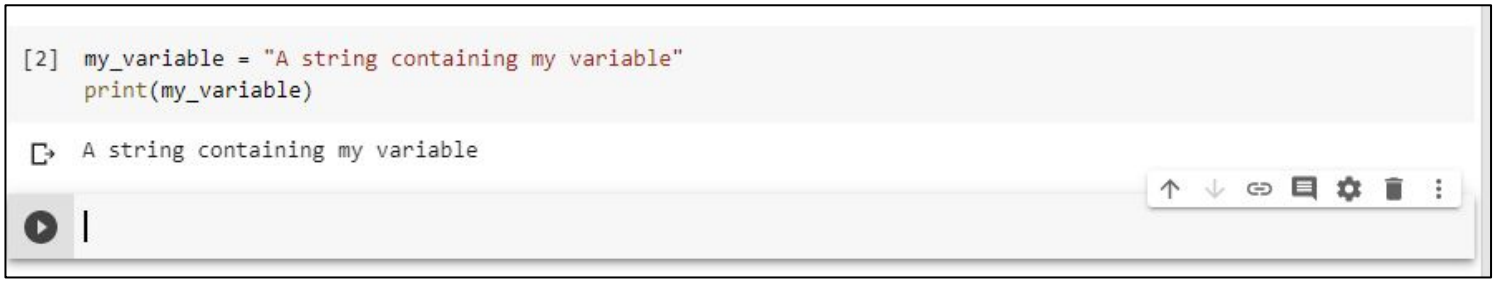
Python Syntax - Variables and Identifiers

In python, variables are assigned using the following syntax:

```
my_variable = "A string containing my variable"
```

Here, I have assigned the phrase `"A string containing my variable"` to the variable `my_variable`

Can I use a variable with a print function?



Yes!

Assigning different variables

- Variable names can begin with a letter or an underscore, and can be followed by any number of letters, underscores and numbers.
- It cannot contain any spaces, and may not begin with a number.
- Finally, there are a list of reserved words in python that cannot be used as user-defined variables
- In most text editors, these will appear to be a different colour, so you'll easily be able to identify them!
- Finally, note that python is case-sensitive:
  - **A**ndy is not the same as **a**ndy

Forbidden variable names

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

https://www.tutorialspoint.com/python/python_basic_syntax.htm

Variable Names - Good or bad?

Variable	Good or Bad?	Reason
my_variable		
my variable		
exec		
_exec		
x9		
9x		

What can be assigned to a variable? Data Types

There are several built-in data types in python:

- String
- Number
  - Integer
  - Float
  - Long
  - Complex
- List
- Tuple
- Dictionary

Python Strings

- We already used these! They are identified as a series of alphanumeric characters between two double quotation marks or single quotation marks:
  - “String1” is the same as ‘String1’
- Strings can be added together, and individual letters/numbers of the string can be accessed through *indexing*.
- Indexing a string is done as follows:
- **String***[n]* returns the n^{th}+1 character in the string
- Let’s walk through an example.

Adding and indexing Strings

```
[6]  string1 = "This is a string"

     string2 = "And this is also a string"

     print(string1)
     print(string2)
     print(string1 + ' '+ string2)

     new_string = string1 + ' ' +string2

     print ("The 12th letter of the new string is " + "'" + new_string[11] + "'")
```

Indexing! This finds the (11+1) letter of the string

```
This is a string
And this is also a string
This is a string And this is also a string
The 12th letter of the new string is 't'
```

Getting a substring from a string - splicing

Sometimes, you will want to search a phrase for a particular identifier. For example, what if I want the name of a file without the extension? To get multiple letters in a row, use a colon, **:**, to separate the start and end of the substring

```
[15] filename = 'CV_Methanol_CU_catalysis2020.csv'
     print((filename[0:11]))

     CV_Methanol
```

Multi-Indexing! This takes all letters in the string from the 0th (inclusive) to the 11th (exclusive), for 11 total letters

- Note that python starts at index 0 not 1 → [0:n] will take the first n digits, because it takes the 0th, 1st… (n-1)th and NOT n.
- However, this doesn't help us if we don't know how long the filename is, and is not very general.
- We can use the **len()** function to get the length (number of characters) of a string.

Getting a substring from a string - splicing

Use the len function to get just the filename and just the extension!

```
number_letters = len(filename)
extension = filename[number_letters - 4:]
name_only = filename[0:number_letters - 4]

print(extension)
print(name_only)
```

```
.csv
CV_Methanol_CU_catalysis2020
```

If no number is present after the colon, it returns everything left in the string. If no number is present before a colon, it returns everything up to the number after the colon

Indexing also “wraps around” string: [-1] is the last letter in the string, [-2] is the second last, etc...

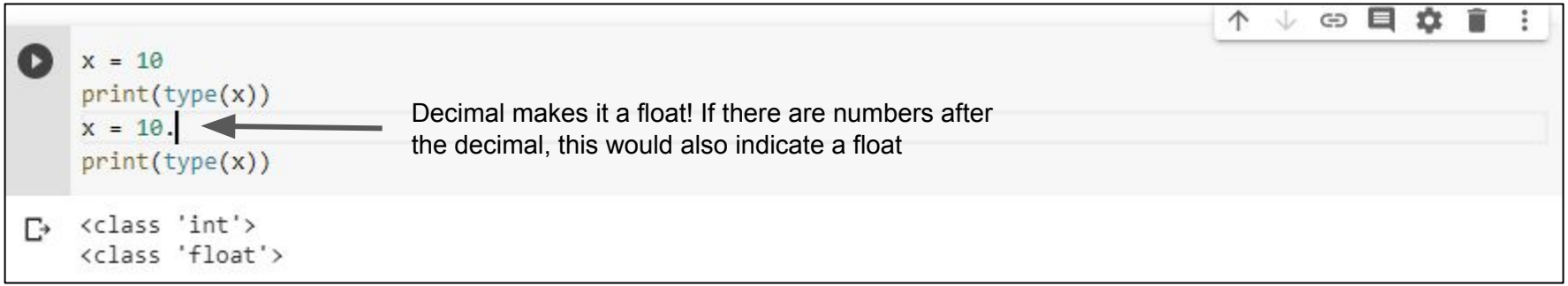
```
[19] print(filename[-1], filename[-2], filename[-3], filename[-4])
```

```
v s c .
```

Try finding the name_only and extension using “negative” indexing

The next Datatype - Numbers

The two primary number Datatypes you will encounter are **integers** and **floats**.
Just think of them as rounded, and not rounded.



In Python 3.x, when two integers are divided, the result is by default a float. This is perhaps counterintuitive, and should be considered when developing own code.
Your data input into ML models will be floats; integers will be used in more general programming

The next Datatype: Lists

- Lists are extremely useful and versatile data structures!
- They can be thought of as:
  - A container in which you can place other datatypes
  - A 1D vector (Matlab)
  - A column (Excel)
- Unlike Matlab vectors, a **single list can contain different data types**
- Lists are indexed with the same syntax we learned for strings
- Lists can be dynamically updated and do not need to have their size pre-determined
- Lists are an excellent way to represent XY data→ one list can contain the X data, and another list can contain the Y data
  - Note: we will almost ubiquitously use Numpy arrays, which will be introduced later. These can be thought of as math-optimized lists, but share many of the same features!

Defining a list in Python

Lists are defined by placing the entries into a list between closed brackets:

```
my_list = ['string1', 10, 'string2', 10.,]
```

In this case, we have created a list that has four entries: two strings, one integer and one float. We can access these entries using the same indexing we learned with strings!

```
[11] my_list = ['sargent',100, 'sinton', 1000., ]
     print(my_list[0])
     print(my_list[2])
     print(my_list[1:3])
```

Note that this will return a list

```
sargent
sinton
[100, 'sinton']
```

Reassigning lists

List entries can be changed by setting the position of the list equal to a new entry:

```
my_list[1] = "A new entry"
```

This can be done when assigning multiple entries as well, as long as the lists are the same size:

```
my_list[0:4] = ["A new entry", 10, "a new entry2", 30]
```

Length = 4 Length = 4

```
[15] my_list = ['sargent',100, 'sinton', 1000., ]
     my_list[3] = 2000.
     print (my_list)
     my_list[0:3] = ['sinton',0.82, 'sargent']
     print (my_list)

     ['sargent', 100, 'sinton', 2000.0]
     ['sinton', 0.82, 'sargent', 2000.0]
```

Where might this be useful?

- Setting data before the experiment had started to 0 (in a time-series)
- Using the first N points to calculate a background which can then be subtracted from the rest of the data (list)
- Fitting data to exponential-decay, but only fitting decay to a certain point
- Lots of other cases!

Editing lists with built-in tools: methods

- Lists can be edited using a few different methods:
  - ***append*** (element): add *element* to the end of a list
  - ***insert*** (index, element): insert a new *element* at *index*
  - ***pop*** (index): delete the entry at *index*

```
[13] my_list = ['sargent',100, 'sinton', 1000., ]
     my_list.append('lab')
     print (my_list)
     my_list.insert(0, 'ML_tutorial')
     print (my_list)
     my_list.pop(1)
     print (my_list)

     ['sargent', 100, 'sinton', 1000.0, 'lab']
     ['ML_tutorial', 'sargent', 100, 'sinton', 1000.0, 'lab']
     ['ML_tutorial', 100, 'sinton', 1000.0, 'lab']
```

What happened here? We called a ***method*** of the ***list*** class. We will talk more about methods, but objects (such as a *string*, an *integer* or a *list*) have functions that are called by typing ".function_name()" after an object

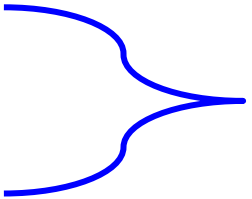
The second-to-last data type: tuples

- Tuples are very similar to lists, except that they are defined using open brackets rather than closed brackets:

```
my_tuple = ('data_point1',2.5, 2, 'data_point2')
```

- Indexing tuples is identical to lists
- The biggest difference between lists and tuples is that **tuples cannot be edited**. They are kind of like “read-only” lists.

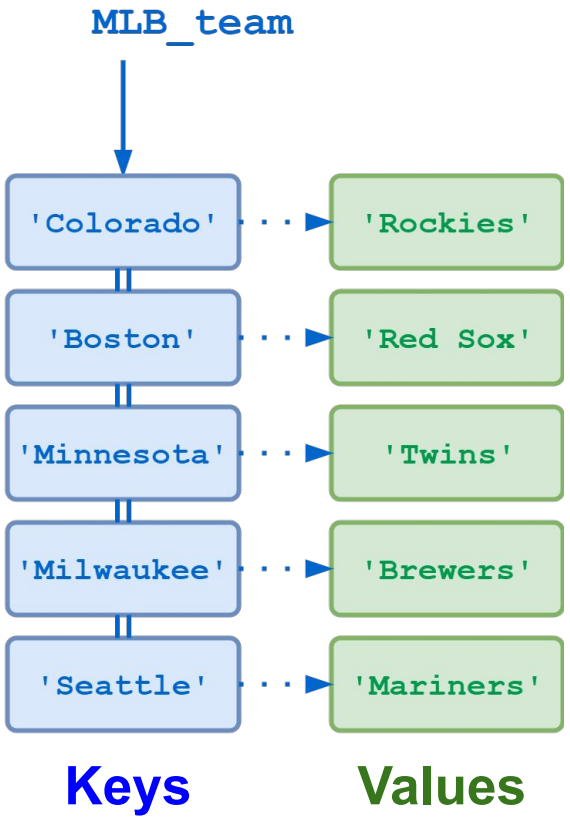
```
[17] my_tuple = ('sargent',100, 'sinton', 1000., )
     print (my_tuple[2])
     print (my_tuple[1:3])
     my_tuple[2] = 100 ##Will throw an error

     sinton
     (100, 'sinton')
     ---------------------------------------------------------------------------
     TypeError                                 Traceback (most recent call last)
     <ipython-input-17-e58676d6ee66> in <module>()
           2 print (my_tuple[2])
           3 print (my_tuple[1:3])
     ----> 4 my_tuple[2] = 100 ##Will throw an error

     TypeError: 'tuple' object does not support item assignment
```



This is an error message! The arrows point to the line that had the error (my_tuple[2] = 100), and then displays what the error caused by a given line was (TypeError). You will see lots of these!

The last datatype (for now): dictionaries!

- Dictionaries are kind of like a table, in which each “key” has an associated “value”.
- This is called a dictionary because in a dictionary you look up a “key” (word) to see its “value” (definition).
- In the example to the right, if we looked up ‘Colorado’ in the MLB_team dictionary, we would receive “Rockies” as the value



https://realpython.com/python-dicts/

Creating Dictionaries in Python

- Dictionaries are created using **{}**. A dictionary can have unlimited entries, but each entry must be a **{***key***:***value***}** pair.
- The *key* needs to be *hashable* - don't worry too much about this, what you should know is that it needs to be a *string*, *number*, or *tuple*. It **cannot** be a list
- The value can be anything you want! String, number, list, tuple, dictionary, whatever!

```
[5]  my_dict = { 'string_key': 1000, 15: [1,2,3], 15.2: 'float_key', (1,'2',3.): 'tuple_key'}
     my_dict = { 'string_key': 1000, 15: [1,2,3], 15.2: 'float_key', [1,'2',3.]: 'list_key'} ##not allowed
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-15b4e2513d57> in <module>()
      1 my_dict = { 'string_key': 1000, 15: [1,2,3], 15.2: 'float_key', (1,'2',3.): 'tuple_key'}
----> 2 my_dict = { 'string_key': 1000, 15: [1,2,3], 15.2: 'float_key', [1,'2',3.]: 'list_key'} ##not allowed

TypeError: unhashable type: 'list'
```

Let's look at that again:

```
my_dict = { 'string_key': 1000, 15: [1,2,3], 15.2: 'float_key', (1,'2',3.): 'tuple_key'}
```

- What are the keys?
  - `'string_key'` (type: str)
  - 15 (type: int)
  - 15.2 (type: float)
  - (1, '2', 3.) (type: tuple)

- What are the values?
  - 1000 (type: int)
  - [1,2,3] (type: list)
  - `'float_key'` (type: str)
  - `'tuple_key'` (type: str)

- Can we check these for some unknown dictionary? Of course!

Referencing Values of Dictionaries

Dictionary values belonging to a given key are referenced with the following syntax:

value = dictionary[key]

```
[4]  print(my_dict['string_key'])
     print (my_dict[15.2])
     print(my_dict[(1,'2',3.)])

     1000
     float_key
     tuple_key
```

Useful Dictionary Methods

```
[9]  ##list all keys of a dicitonary
     print(my_dict.keys()) ##python 2
     print(list(my_dict.keys())) ##python 3

     ##list values of a dictionary
     print(list(my_dict.values()))
```

```
dict_keys(['string_key', 15, 15.2, (1, '2', 3.0)])
['string_key', 15, 15.2, (1, '2', 3.0)]
[1000, [1, 2, 3], 'float_key', 'tuple_key']
```

- Several notes from the above example:
  - We used *methods* - *keys()* and *values()* - of the dictionary *my_dict* to list the keys and values, respectively
  - We then converted the output of those *methods* to a list by calling the *list()* function.
  - This leads into our next topic...

Where are dictionaries useful?

- They are extremely powerful as you can index different datatypes with a name.
- For example, if you have several measurements on the same sample, you can store and reference the data within a code with something like the following:

```
data = {
        'Sample1': {
                'Exp1': Data,
                 'Exp2': Data,
                 },
         'Sample2': {
                'Exp1': Data,
                 'Exp2': Data,
                }
        }
```

A dictionary of dictionaries! Each Sample has several experiments and each experiment has Data. The beauty of this is that if you don't have a certain experiment for a certain Sample, you can just leave that entry blank, and the dictionary will still work! These can be referenced the same way:

exp1_sample1_data = data['Sample1']['Exp1']

Transforming Data

- A datatype can be transformed into a different datatype by calling one of python's built in **functions**:
  - int(x) will convert x to an integer, if possible
  - str(x) will convert x to a string
  - float(x) will convert x to a string, if possible
  - list(x) will convert x to a list
  - There are lots more, but these are the most commonly used ones

- There are some special considerations and other ways to do this, but this will be enough to get you started!

Python Operators

- Python has several built in tools that can be used to operate on different data. For a comprehensive list, see:
  https://www.tutorialspoint.com/python/python_basic_operators.htm
- Some of the more useful ones for math:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Modulus (%)
  - Exponent (**)

```
[14] x = 10.
     y = 2.

     print (x+y) ##should be 12
     print (x/2) ##should be 5
     print (x-y) ##should be 8
     print (x%y) ##should be 0
     print (x**y) ##should be 100
```

```
12.0
5.0
8.0
0.0
100.0
```

Python Logic Operators

a = 10, b = 20

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	
<	If the value of left operand is less than the value of right operand, then condition becomes true.	
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	

Logic Operators are used to compare two (or more) things. These are used in **if statements**!

https://www.tutorialspoint.com/python/python_basic_operators.htm

If Statements

In Python, If Statements are used to check if something is true or not, and to do something if it is. The general syntax is:

```
if condition:
    ##do things that you want if the condition was satisfied
elif condition_2:
    ##do something else if a different condition is met
else:
    ##if none of the above conditions were met, do this thing
```

Where *condition* and *condition_2* are evaluated using the operators from the previous slide

Walking Through an Example

```
[19] ##Don't worry about this - just used to generate a random number between 1 and 10
     ##Do note that we are using the int() function though!
     import random
     Andy = int(random.random()*10)
     Hitarth = int(random.random()*10)

     #Check to see what the value is
     print("Andy is a ", Andy)
     print("Hitarth is a ", Hitarth)

     #This means we are checking to see if the value assigned to Andy is less than the
     #value assigned to Hitarth. This is equivalent to saying " if Hitarth > Andy"
     if Andy < Hitarth:
         #If and only if Andy is < Hitarth, do the following
         print ("Questionable taste!")

     #No, only if Andy WAS NOT less than Hitarth, check to see if they are identical
     elif Andy == Hitarth:
         #If Andy and Hitarth are the EXACTLY the same value, do the following
         print ("What are the chances of that!")

     #Now, if NONE of the above conditions were met, i.e Andy is > than Hitarth, enter
     #The else loop
     else:
         print ("Nice")
```

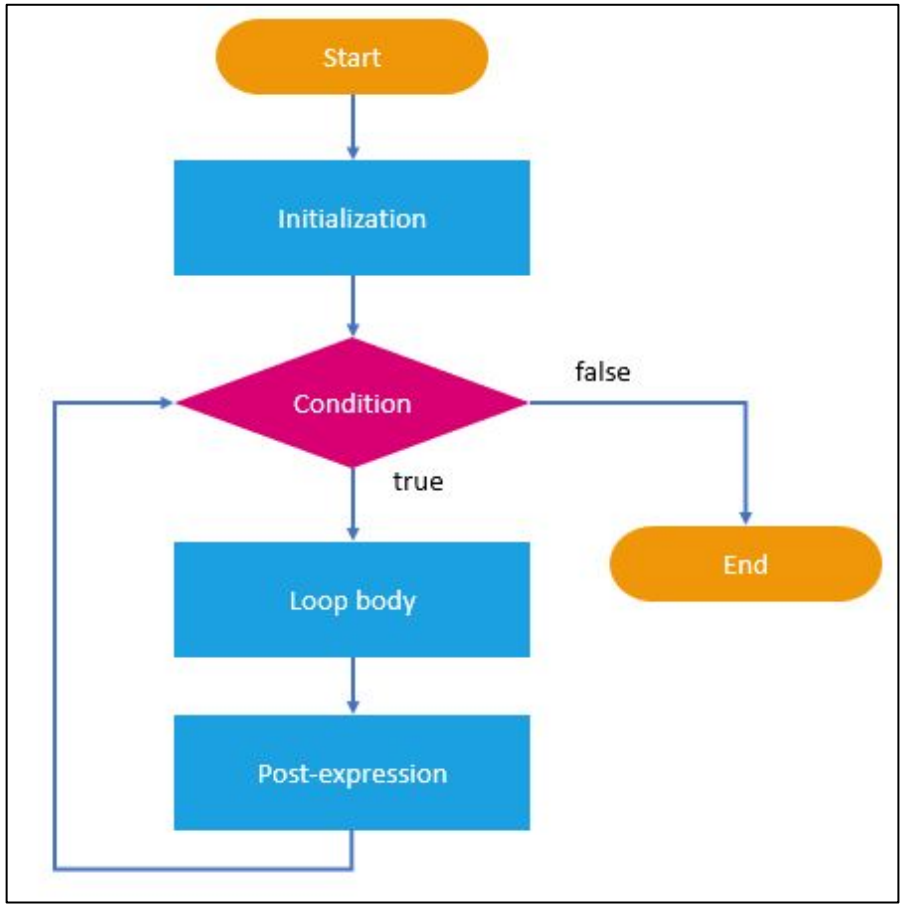
```
Andy is a  5
Hitarth is a  7
Questionable taste!
```

Programming Loops

Loops are one of the most important concepts in programming.

The general syntax for a loop is shown to the right.

Effectively, you check a *condition*, and if this condition is satisfied, you execute some code. This code will do some stuff, and then you go back and check the *condition* again. This continues (or *loops*) until the *condition* **is no longer met**



https://www.javascripttutorial.net/javascript-for-loop/

Loops in Python

- Python has two types of loops:
  - *While loops*
  - *For loops*

- These are extremely important and you will use these all the time in your own code.

- We are going to walk through each of these, and describe how they work

While Loop

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

```
[1]  ##this is some counter
     i = 0

     ##i < 10 is the condition
     while i < 10:
         print ("We are counting, and are at " + str(i))
         ##we need to update the counter, otherwise i will ALWAYS be less than 10 and the loop will never end!
         i = i+1
```

```
We are counting, and are at 0
We are counting, and are at 1
We are counting, and are at 2
We are counting, and are at 3
We are counting, and are at 4
We are counting, and are at 5
We are counting, and are at 6
We are counting, and are at 7
We are counting, and are at 8
We are counting, and are at 9
```

For Loop

- I personally use these far more often than while loops
- In a *for* loop, you iterate through an *iterable* (very commonly a *list*)
- The general syntax is:

```
For item in a_list:
    #do something to item
```

```
[3] my_list= [0,1,2,3,4,5,6,7,8,9]
    for num in my_list:
        print("The next number in the list is" + str(num))

    The next number in the list is0
    The next number in the list is1
    The next number in the list is2
    The next number in the list is3
    The next number in the list is4
    The next number in the list is5
    The next number in the list is6
    The next number in the list is7
    The next number in the list is8
    The next number in the list is9
```

```
[4] my_other_list = [0, 'A string', (0,1,2), 'Another String']
    for item in my_other_list:
        print ("The next item in the list is", item)

    The next item in the list is 0
    The next item in the list is A string
    The next item in the list is (0, 1, 2)
    The next item in the list is Another String
```

- The list can be whatever you want!

A useful function: range

- Often, you just want to increase a counter to some fixed number (i.e, try up to 10000 iterations of training an ML model)
- In this case, you can use a built-in function called *range* to generate a list of numbers that increases by 1 every index.

```
[8] a = range(0,10)
    print(a)
    for i in a:
        print(i)

    range(0, 10)
    0
    1
    2
    3
    4
    5
    6
    7
    8
    9
```

This is just a list that runs from 0-->9

When are loops useful?

- All the time!
- Particularly for iterating over different files to extract data
- Some data manipulation is done with for loops
- Iterating over hyperparameters in a ML model
- Anything else that requires iteration

Defining Your own functions in Python

- You may wish to define a function if you need to execute the same task multiple times:
  - Formatting graphs
  - Normalizing data
  - Fitting data
  - others

- Defining a function in Python is straight-forward:

```
def function_name(inputs):
    #do some stuff
    return any_outputs
```

Let's make a function, and then call it!

First, we define a function with a given name:
print_a_word_function

We then list the *inputs* needed to call the function between brackets:
(phrase_to_print)

We then have our function print the *input* it is given:
print *(phrase_to_print)*

We then call the function, passing in the (universally true) phrase: "Andy is awesome!"

```
[10] def print_a_word_function(phrase_to_print):
         print (phrase_to_print)

     print_a_word_function('Andy is awesome!')

     Andy is awesome!
```

Some notes to consider with functions

- Variables created within a function are only real within that function
- Default values can be given to inputs, which then don't need to be defined
- If you don't pass the right number of mandatory inputs, you will get an error
- If one of the inputs you pass is the wrong type, you will get an error
- Returning an output is not required!

```
## show some function stuff
##c and d are optional parameters - they will be assigned 2 and 4
##if they are not specified. a and b are required!
def my_function(a, b , c = 2, d = 4):
    y = a*b+c/d
    return y

##we can assign the output of a function to a variable
test1 = my_function(1,2,3,4)
##can define variables out of order (c and d are optional)
test2 = my_function(a = 1, c = 3, b = 4, )

print('The value of test1 is ', test1)
print('The value of test2 is ', test2)

##uncomment the following to try!
########Errors
## not enough inputs
#test4 = my_function(10)
##wrong input type
#test3 = my_function(a = 10, b = 'str')
##y is not defined, not a global variable
#print(y)
```

```
The value of test1 is  2.75
The value of test2 is  4.75
```

The difference between functions and methods

- Functions can just be called on their own
  - They are called by typing *output = function(input)*
- Methods are functions that belong to a specific instance of a class.
  - They are called by typing *output = instance.method(input)*
  - They often modify the original *instance*, but can still be assigned to an output

```
[16] ##difference between function and method

     my_list = [10, 5, 100, 20, -10]
     print ("The default list is ", my_list)
     ##sort the list using the built in sorted function
     new_list = sorted(my_list)
     print ("The new list created with sorting function is ", new_list)
     ##modify the existing list using the sort method
     my_list.sort()
     print ("The old list has been modified with the sort method and is now ", my_list)

     The default list is  [10, 5, 100, 20, -10]
     The new list created with sorting function is  [-10, 5, 10, 20, 100]
     The old list has been modified with the sort method and is now  [-10, 5, 10, 20, 100]
```

Today's Agenda

- Opening files with python
- Extracting data and formatting it
- Importing and using python modules
- Essential python modules:
  - Numpy
  - Matplotlib
  - Scipy
  - Re
- BREAK
- Pandas + Formatting Data for ML (Hitarth)

Reading and writing files in Python

- This is an essential consideration for any scientist using Python! You need to open files to read data to analyze.
- Typically, you use the ***os*** module to access different file systems on your computers
- In google colab, you must first mount your drive into the runtime:

```
from google.colab import drive
drive.mount('/content/drive')
```

- This will prompt you to go to a URL, login with your google account and paste an authorization code. Afterwards, it should say:

```
Enter your authorization code:
··········
Mounted at /content/drive
```

- You already did this earlier! But I include this just for reference

Opening and Reading a file

- The syntax is:
- `f = open(filename, 'access')`
- Filename is the path (remember those?), either relative or absolute, to the file
- `'access'` is typically *'r'* for *read*, or *'w'* for *write*
- Calling f.read() will print the contents of the file pointed to by filename

```
filename = '/content/drive/My Drive/Doctorate Studies/ML Course/ML_tutorial_data.txt'
f = open(filename, 'r')
print (f.read())
f.close()
##uncommenting the below will throw an erro--> f is now closed!
#print(f.read())

##rather than manually closing files, you can use with:
##outside of the indent block, f is automatically closed!
with open(filename, 'r') as f:
    print(f.read())
```

```
# Wavelengths, Counts
3.9000000000000000e+02 1.5940000000000000e+03
3.9200000000000000e+02 1.5800000000000000e+03
3.9400000000000000e+02 1.5840000000000000e+03
3.9600000000000000e+02 1.6480000000000000e+03
3.9800000000000000e+02 1.6320000000000000e+03
4.0000000000000000e+02 1.6700000000000000e+03
4.0200000000000000e+02 1.5960000000000000e+03
4.0400000000000000e+02 1.5780000000000000e+03
4.0600000000000000e+02 1.5940000000000000e+03
4.0800000000000000e+02 1.6800000000000000e+03
4.1000000000000000e+02 1.5460000000000000e+03
4.1200000000000000e+02 1.6280000000000000e+03
4.1400000000000000e+02 1.6020000000000000e+03
4.1600000000000000e+02 1.7240000000000000e+03
4.1800000000000000e+02 1.5860000000000000e+03
4.2000000000000000e+02 1.7060000000000000e+03
4.2200000000000000e+02 1.6040000000000000e+03
4.2400000000000000e+02 1.6300000000000000e+03
```

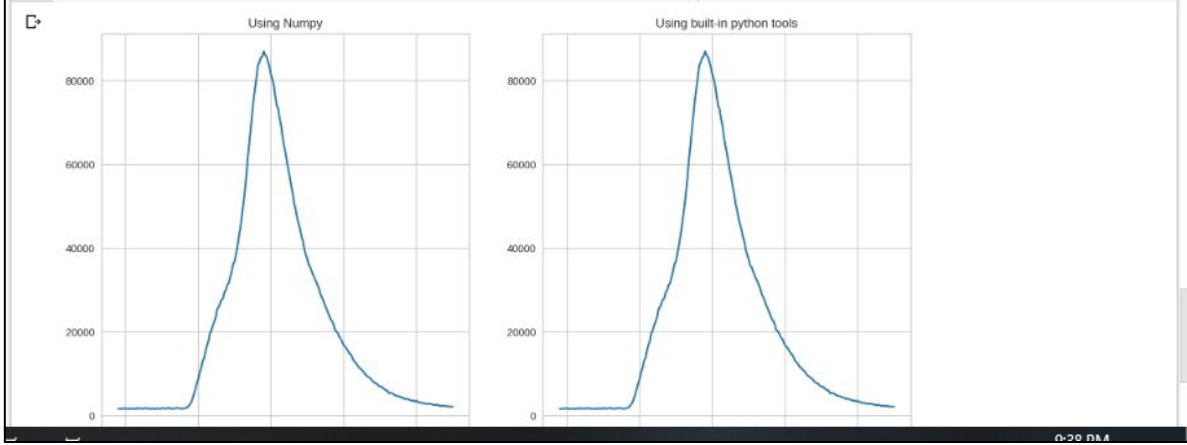
Extracting the relevant data from a file

- Many different ways to do this, which will be discussed more in the Data Formatting Section (tomorrow)
- Depends on the filetype you are loading
- Some examples on next slide

Making sense of data

- In general, requires knowledge of what the file contents look like
- The good news is that you can usually figure this out using the “print” statement.
- It is much easier to do this with libraries (like numpy) than with built-in python functions
- Take some time later and try and understand what this is doing (and see assignment!)

```
##used to plot, ignore for now
%matplotlib inline
import matplotlib.pyplot as plt
f,ax  = plt.subplots(nrows = 1, ncols = 2, figsize = [14,7])
##using numpy
with open(filename, 'r') as f:
    data = np.loadtxt(f)

ax[0].plot(data[:,0], data[:,1])
ax[0].set_title('Using Numpy')
##using built in file object methods
with open(filename, 'r') as f:
    data = f.read()
##convert a string to a list, separating on each new line ('\n')
list_data = data.split('\n')
##create two empty lists for wavelength and counts
w = []
c = []
##iterate over each line in the list which contains a string. skip the first line! it is a header.
##also skip the last line! it is an empty line. use indexing!
for line in list_data[1:-1]:
    ##the string looks like: 'wavelength counts'
    ##which means we can separate the string using a blank space
    w.append(float(line.split(' ')[0]))
    c.append(float(line.split(' ')[1]))
ax[1].plot(w,c)
ax[1].set_title('Using built-in python tools')
plt.show()
```



Writing Data

- Use a module to write data (pickle, numpy, pandas).... Will be covered in upcoming slides!

Python Modules

- With some basic understanding of Python, it is now time to understand what really makes python useful: Modules

- Python is open-sourced, which means that everyone around the world with an internet connection can contribute to its development (kind of)

- A module contains code that can be imported and used by your own python files

- **If you are looking to do something, chances are there exists somewhere a library where someone has done it better than you will ever be able to :)**

List of Modules

- In Google Colab, typing “ `help('modules')` ” will print a list of all of the modules that are available in your current environment.
  - Warning: There are a lot. Don’t worry, nobody knows what they all do
- It will have almost everything you need by default, but it is possible to install additional ones (not covered here, we can address it as the need arises)
- We will go over a few of the important modules, but not in a ton of depth - you will need to explore them on your own!

How to import modules

- The general syntax is:

```
import module
```

- After this, you call different functions and classes from the module as if they were methods from the module:

```
output =  module.function(inputs)
```

The “*import as*” syntax

- It is sometimes better to abbreviate the module name, as the module name may be too long

```
[19] ##import the numpy module with the name np
     ##Prelude of things to come :O
     import numpy as np

     ##can now call different numpy functions by using the name "np"
     x=np.arange(0,10)
     print (x)

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

Andy's list of essential modules for data processing

- numpy (https://numpy.org/)
  - Used to convert lists into MatLab-like vectors. Really easy to do things like normalize data, fit data, interpolation, ALL vector and matrix math, lots more
- Scipy
  - In conjunction with numpy, very useful for fitting functions, some standard data analysis
- pandas (https://pandas.pydata.org/docs/user_guide/index.html)
  - Used to manage and visualize large sets of data. Extremely useful in machine learning, and is kind of like python's version of Excel
- re (https://docs.python.org/2/library/re.html)
  - Regular Expression manager, **extremely useful** for parsing filenames quickly and robustly
- os (https://docs.python.org/3/library/os.html)
  - Necessary to navigate through different folders in search of all of your data
- matplotlib (https://matplotlib.org/)
  - Plotting. It is almost the same as plotting in MatLab, but can also be tricky to master if you are trying to do fancy things
- pickle (https://docs.python.org/3/library/pickle.html)

Numpy

- Numpy's primary use is to take a list of numbers and transform them into an *array*.
- An *array* is a vector. It can be multidimensional (like a matrix!) and is easy to index.
  - Indexing is similar to Matlab: array[i,j] returns the value at row (i+1) and column (j+1).
- Element-wise AND matrix operations can be performed on arrays, unlike lists, and as a result they are far more efficient to analyze data!
- What is an element-wise operation?
  - You perform some operation on each item in an array
  - This is the default for numpy, which is in contrast to Matlab

Usefulness of Numpy

Consider the case of wanting to multiply your data by 2:

```
[9]  ##Compare numpy to lists
     import numpy as np

     my_list = [0,1,2,3,4,5]
     my_array = np.array(my_list) ## pass a list to np.array() to make a numpy array
     print (my_list*2)
     print (my_array*2)
```

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

Lists in python will just be duplicated! Numpy arrays will actually be multiplied
All math operations can be performed on NUmpy arrays: Note that when you add (or subtract) two vectors, **they must be the same length**

Useful methods/attributes of numpy arrays

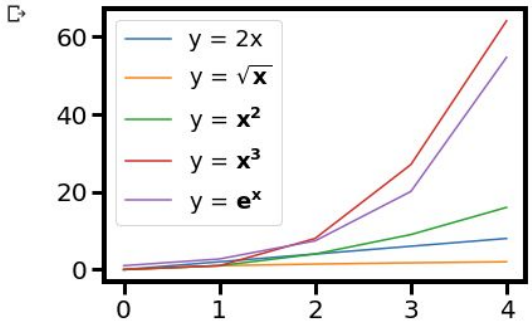
Full list here:https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

- argmax: return index of maximum value in array
- argmin: return index of minimum value in array
- flatten: collapse array into 1-dimension (useful for ML!)
- nonzero: return indices of all non-zero elements
- sort: sort array from highest to lowest

- The most useful aspect of numpy arrays is that they can be operated on with mathematical operators

Numpy examples continued

```
[8] import matplotlib.pyplot as plt
    x = np.arange(0,5, dtype = float) ##can specify the type of number in the array
    y1 = 2*x
    y2 = np.sqrt(x)
    y3 = x**2
    y4 = x**3
    y5 = np.exp(x)
    list_of_names = [r'y = 2x',r'y = $\bf{\sqrt{x}}$',r'y = $\bf{x^2}$',r'y = $\bf{x^3}$',r'y = $\bf{e^x}$']

    ##let's plot some different functions with numpy!
    f, ax = plt.subplots(nrows = 1, ncols = 1)

    for i, data in enumerate([y1,y2,y3,y4,y5]):
        ax.plot(x, data, label = list_of_names[i])

    ax.tick_params(axis='x',labelsize = 20, width = 3,length =10)
    ax.tick_params(axis='y',labelsize = 20, width = 3,length =10)
    [i.set_linewidth(4) for i in ax.spines.values()]
    h, l = ax.get_legend_handles_labels()
    ax.legend(h,l, fontsize = 18, loc = 'best')

    plt.show()
```



- Can easily apply different functions to a numpy array, and can make your own!
- Super necessary to take raw data and convert it into something meaningful and useful
- There are other uses for numpy, but in general you use them to perform any math you need to do on data
- These becomes extremely powerful when you have more than one dimension in your data: something like a 2D-dataset (TA data, AFM images, spectrally resolved PL decay, etc.) where each value is at a (i,j) coordinate

Numpy Summary

- You will use this library all the time!
- Practice with it, and actively try to use this to manipulate your experimental data.
- Even if you only have to make one graph, try loading and plotting the data using numpy + matplotlib

Today’s Agenda

- Matplotlib Example
- Introduction to the os, pickle and scipy modules (will revisit at the end if time)
- Using ML to guide experiments:
  - Manually predictions
  - Computational predictions
- General Q/A period

The ***matplotlib*** module

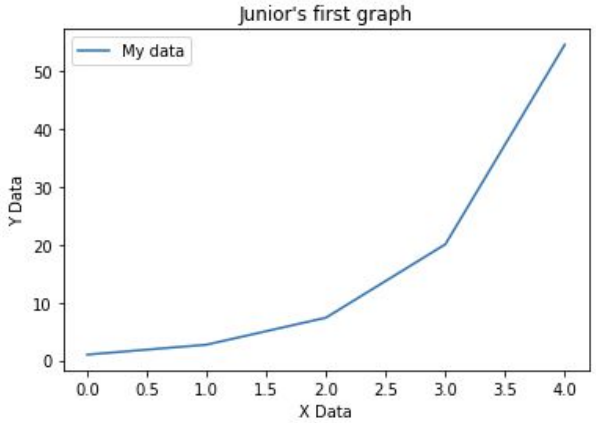
- It is a library to plot data, and the plots and syntax are designed to look like Matlab.
- It can be kind of finicky, so I recommend that once you have made a few graphs that look nice you save the code in a function that you can import and use!
- We will cover the general use case here - in fact we have already used it!

```
f, ax = plt.subplots(nrows = n, ncols = n)
ax.plot(DATA)
f.savefig('name')
```

Here’s an example!

We generate some data, plot it and do general formatting to the plot!

```
##import the pyplot functions of matplotlib
#ask it to show the plots in jupyter notebook
import matplotlib.pyplot as plt
%matplotlib inline
##generate some data
x = np.arange(0,5, dtype = float)
y = np.exp(x)

##create a figure (f) and an axes (ax)
f, ax = plt.subplots(nrows = 1, ncols = 1)
##plot data!
ax.plot(x,y, label = 'My data')
ax.set_xlabel('X Data')
ax.set_ylabel('Y Data')
ax.set_title("Junior's first graph")
#get legend data (defined with "label" for each data plotted")
h, l = ax.get_legend_handles_labels()
ax.legend(h,l)
plt.show()
```



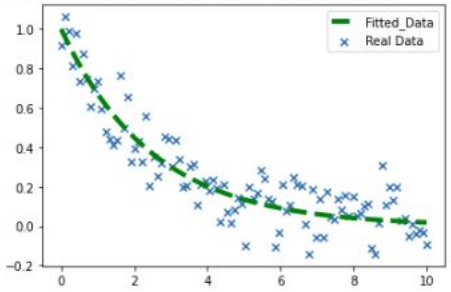
The ***scipy*** module

- I primarily use this for fitting and smoothing data
- Has a “curve_fit” function, which is similar to Matlab’s
- Let’s walk through an example - you will need the “Noisy_Exp.txt” which has been uploaded to the shared google drive folder

Fitting an exponential with ***scipy***

```
[36] ##Load data
     noisy_data = '/content/drive/My Drive/Doctorate Studies/ML Course/Course Material/Noisy_Exp.txt'
     with open (noisy_data, 'r') as f:
         data = np.loadtxt(f)
     x_real = data[:,0]
     y_real = data[:,1]
     ##define function to which we want to fit data
     def single_exp(x, tau):
         y = np.exp(-x/tau)
         return y
     ##import curve_fit (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html)
     ##note that it returns the parameters of the function and their covariance (read docs!)
     from scipy.optimize import curve_fit
     ##fit to x and y dara
     [fit_params, fit_covar] = curve_fit(single_exp, x_real, y_real)
     print ("The fitted value is {0:.2f} with a standard deviation of {1:.2f}".format( fit_params[0], np.sqrt(np.diag(fit_covar))[0]))
     ##calculate the predicted data using the fitted parameter
     y_pred = single_exp(x, fit_params[0])
     ##finally, plot!
     f,ax = plt.subplots(nrows = 1, ncols = 1)
     ax.scatter(x_real,y_real, label = 'Real Data', marker = 'x')
     ax.plot(x_real, y_pred, linestyle = '--', linewidth = 4,color = 'g', label = 'Fitted_Data')
     h, l = ax.get_legend_handles_labels()
     ax.legend(h,l)
```

```
The fitted value is 2.50 with a standard deviation of 0.11
<matplotlib.legend.Legend at 0x7f0274085dd8>
```



- This takes practice! The syntax will generally be the same, but sometimes you will need to pass an array of fitting parameters (if it is a multi-variant fitting process)
- Can get confusing with unpacking variables, and passing dummy functions to allow for non-fitted parameters

The ***os*** module

- Used to navigate through the file system! Can be used to find all files, check if something is a file, etc
- Similar in funtionality to the linux commands we learned earlier in the course

```
[53] import os
     ##check current directory
     print(os.getcwd())
     ##what is here?
     print(os.listdir())
     ##let's check what's in the ML Course drive!
     print(os.listdir('drive/My Drive/Doctorate Studies/ML Course/'))
     wd = 'drive/My Drive/Doctorate Studies/ML Course/'
     for my_file in os.listdir(wd):
         if os.path.isfile(wd + my_file):
             print ("This one is a file:",my_file,"!")
```

```
/content
['.config', 'drive', 'sample_data']
['Online Courses at Sargent U.gsheet', 'Untitled document.gdoc', 'Course Material', 'Working_Python_NoteBook.ipynb', 'Python_Tutorial.ipynb']
This one is a file:  Online Courses at Sargent U.gsheet !
This one is a file:  Untitled document.gdoc !
This one is a file:  Working_Python_NoteBook.ipynb !
This one is a file:  Python_Tutorial.ipynb !
```

The ***pickle*** module

Store your data with pickle. It “serializes” python objects, you can then load your data afterwards. It can be used to save numpy arrays, lists, dictionaries, and pandas dataframes (to be taught).

```
[58] import pickle
     x = np.arange(0,5, dtype = float)
     y = np.exp(x)
     ##saving an object
     pickle.dump(x, open(wd + 'test_x.pickle','wb'))
     pickle.dump(y, open(wd + 'test_y.pickle','wb'))
     x_load = pickle.load(open(wd + 'test_x.pickle','rb'))
     y_load = pickle.load(open(wd + 'test_y.pickle','rb'))

     print (x - x_load)
     print (y - y_load)

     [0. 0. 0. 0. 0.]
     [0. 0. 0. 0. 0.]
```

The ***re*** module

- This is how to implement *regular expressions*.
- Regular expressions are pretty extensive! They allow you to find patterns in strings.
- We are not going to cover regex here since everyone will have a different level of familiarity with them. Just know that in python there is a library that allows you to implement regex.

Moving on to the pandas module

- This is really the first module for which its use is primarily for machine learning.
- Hitarth will walk you through some examples, and will discuss how you can use pandas to load, format and prepare your data for use in machine learning!

Simple ML modelling

Overview

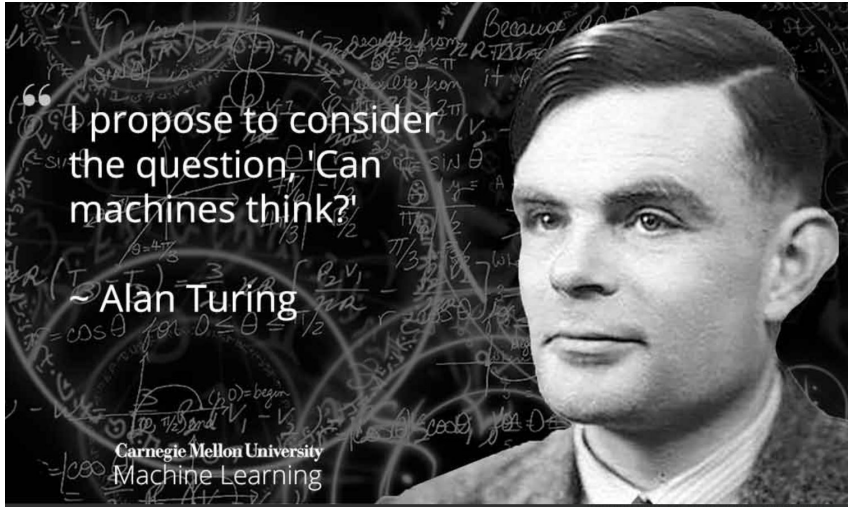
- What is ML?
- The simplest ML model - Linear Regression
- Overview of different models in SKLearn:
  - Naive Bayes
  - Support Vector Machines
  - K-means clustering
  - Decision Tree
  - Random Forest
- Accounting for everything from before - introducing TPot!

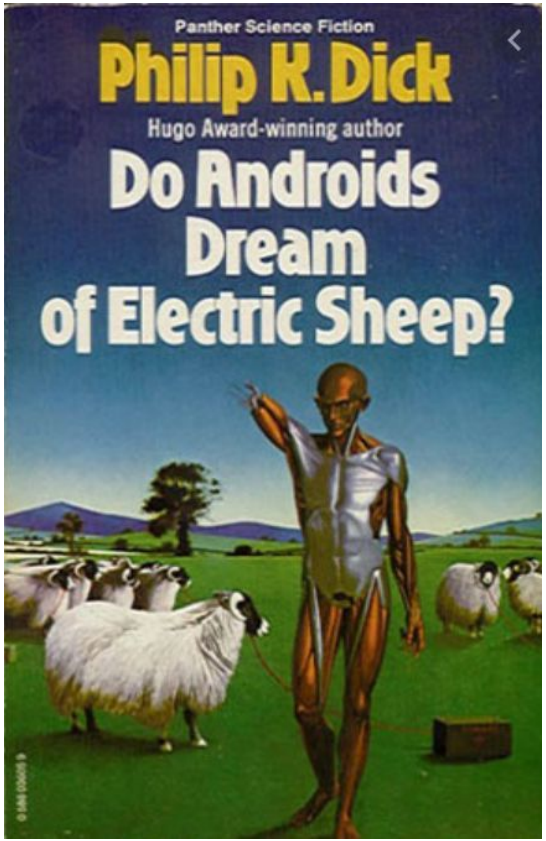
Data Curation, Standardization and Representation

Overview:

- Why does this matter?
- Tips and tricks to standardize data
  - Using pandas!
- Representing experiments - fingerprinting

Artificial Intelligence!





- **Central question**: What's intelligence?
- What's learning?

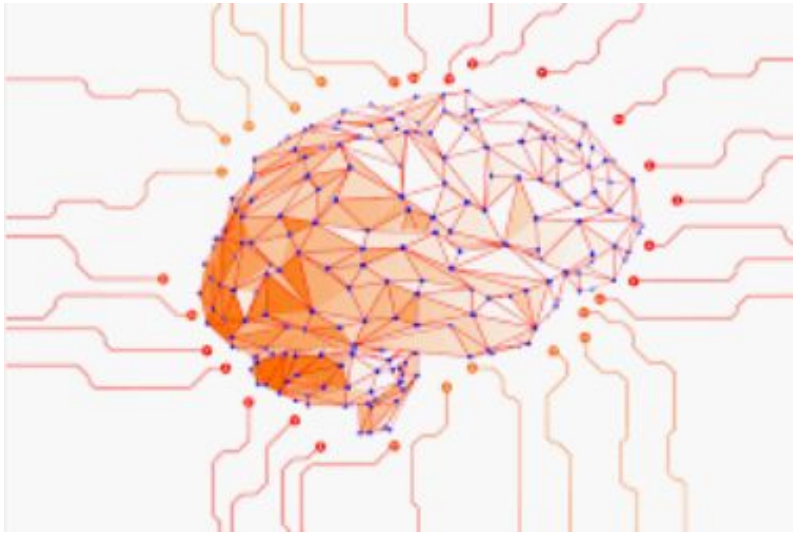
- How do humans learn?

What is Machine Learning?

- Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

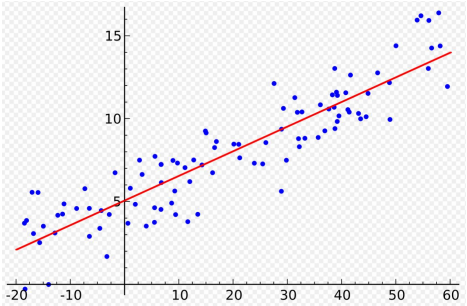
- The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.

- The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.



Different types of ML methods

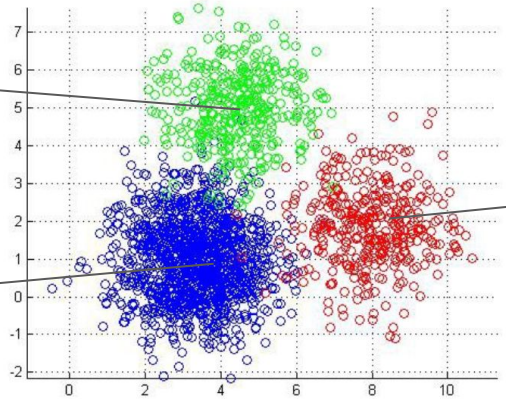
- Supervised Learning:
  - Using labelled datasets, we make the machine learn! Ex: Regression
  - We are given datasets as pairs {data, data_labels}
  - And we want to predict labels on unseen datapoints



- Unsupervised Learning
  - We ask the Machines to learn patterns in our data! Ex: Clustering

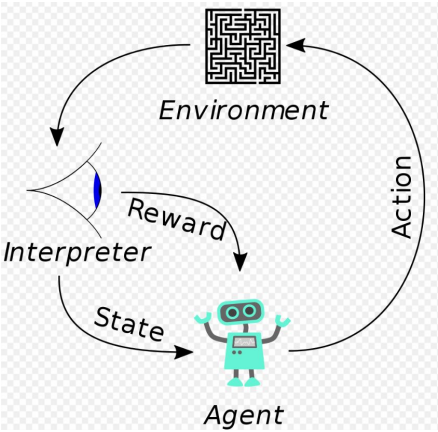
Bandgaps in range 2+

Bandgaps in range 1-2

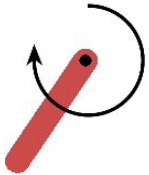


Bandgaps in range 0-1

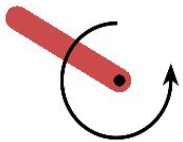
- Reinforcement Learning:
  - Machines decides the best set of steps depending on the reward from the environments!
  - Can a machine learn to keep the pendulum stationary in an unstable equilibrium?



Learning phase of RL!



Machines learn!



Do machines dream? Most will agree to ‘NO’ as an answer
But can they dream? Well, we aren’t sure yet!

Data!!

- Let’s make Machines learn the same way humans learn!
  - Through experience!
- Humans have trained through:
  - Evolutionary experiences
  - Individual lifetime experiences
- Similarly we need data to teach the machines!
  - So, let’s get started with how to deal with data!
  - PANDAS!!

Panel Data(s)?

How to get Data?

There are lots of databases that exist out there. A few of them are experimental and others are based on theoretical calculations like DFT or MD or MCMC. A few of these are:

- Your own experimental data
- Materials Project
- OQMD
- AFLOW
- quantum-machine.org
- NREL database
- 2D materials encylopedia
- Drugbank (COVID19 drug search)!

How do Machines Learn?

So, one must now ask what are the different ways by which we can make Machine Learn?

- There are quite a lot of Supervised Machine Learning algorithms that exist
- All of them, can however, be classified as either based on Deep Learning or not based on Deep Learning
- Deep Learning is a subset of Machine Learning algorithms taking the inspiration from information processing in biological systems
- However, such deep learning based methods require huge amount of datasets and hence, becomes impractical if we have smaller datasets or small amount of data
- Non deep-learning methods do not suffer from this curse! And when used properly are capable of handling complex problems as well.

Data Naming conventions

Before we go ahead and make machines learn, we need to set some conventions that we will follow to represent our data.

Let's say we have N pairs of datapoints: (X_i, y_i) where $0 \leq i \leq N-1$. We can accumulate all the datapoints as a matrix $\mathbf{X}$ and $\mathbf{y}$ such that $\mathbf{X} = [X_0, X_1, .., X_{n-1}]$ and $\mathbf{y} = [y_0, y_1, \ldots, y_{n-1}]$.

The pairs (X_i, y_i) can be anything. They can represent:

- (crystal structure, bandgaps)
- (DNA sequence, selectivity)
- (Images, person's name)
- (Speech, emotion) and so on...

For our little ML class, we will focus on the materials dataset that we downloaded before!

Training and testing dataset

- Once we make our machines learn on data, how can we be sure that they actually learnt anything?
- This is the classical problem of learning v/s memorizing
- Our ML models can simply remember everything we provide and not learn anything at all.
- This will result in very high accuracy for our provided data but they won't perform very good on unseen datasets.
- Thus, what we do is that we set a portion of our data for testing our ML models; we call it **Testing dataset**
- The strategy thus, is that given a dataset tuple (X, y), we split them into two tuples: $(train\_X, train\_y)$ and $(test\_X, test\_y)$

Let's change our working directory

Mount your Google drive first using the intructions shown before in the slides!

- Now go to the directory where you have saved the files we sent you (specifically *CO_CO2_RR_exp_database.xlsx*)

```
[ ]  cd drive/My\ Drive/quantum_mechanics
```

```
/content/drive/My Drive/quantum_mechanics
```

Handling Data (Intro to Pandas)

Functionalities

- We suggest you to save your data in tabular forms as .csv or excel files
- Pandas is a great python library to handle data - both for loading and storing data
- In the following couple of sections of this notebook, we will show how to load, process and save data using Pandas!

Loading data using Pandas

- Let's say your experimental data is present in a tabular form (See the file - CO_CO2_RR_exp_database.xlsx)
- Pandas allows to read files with just one line!

```
df = pd.read_excel('CO_CO2_RR_exp_database.xlsx')
```

- Pandas allows you to read csv files as well!

```
df = pd.read_csv('filename.csv')
```

Let’s open the excel file first!

```
[ ]  # Import the Pandas library
     import pandas as pd

     # Read your excel file
     df = pd.read_excel('CO_CO2_RR_exp_database.xlsx')

     # In case you have a csv file, you can use:
     # df = pd.read_csv('filename.csv')

     print(df)
```

```
          Creator  ...      Total
0       Member 1   ...  14.691853
1       Member 1   ...  18.293114
2       Member 1   ...  19.885077
3       Member 1   ...   5.857601
4       Member 1   ...  22.897618
...          ...   ...        ...
3710    Member 9   ...  68.854736
3711   Member 10   ...  93.800000
3712   Member 10   ...  86.500000
3713   Member 10   ...  80.400000
3714   Member 10   ...  84.000000

[3715 rows x 60 columns]
```

```
        Creator  ...      Total
0      Member 1  ...  14.691853
1      Member 1  ...  18.293114
2      Member 1  ...  19.885077
3      Member 1  ...   5.857601
4      Member 1  ...  22.897618
...         ...  ...        ...
3710   Member 9  ...  68.854736
3711  Member 10  ...  93.800000
3712  Member 10  ...  86.500000
3713  Member 10  ...  80.400000
3714  Member 10  ...  84.000000

[3715 rows x 60 columns]
```

- A dataframe object has rows and columns indexed by what we call 'index' and 'columns'
- In the above dataframe, there are 3715 indices and 60 columns. Columns represent different features. Entries in the first column of a csv file or excel file are treated as names of the columns by the read function of Pandas.
- Indices are inferred from row numbers of excel file or csv file.
- Let's dig a bit into how values are being managed in the csv file.

Processing data using Pandas

- You can view specific columns of Pandas dataframe. Let's say we want to view the columns - Total Gas and n-PrOH

```
print(df[['Total Gas','n-PrOH']])
```

```
      Total Gas     n-PrOH
0     14.691853        NaN
1     18.293114        NaN
2     19.885077        NaN
3      5.857601        NaN
4     22.897618        NaN
...         ...        ...
3710  45.115029  11.223501
3711  64.100000   7.900000
3712  52.900000   5.700000
3713  47.600000  10.900000
3714  44.200000  24.000000

[3715 rows x 2 columns]
```

- You can add as many columns name as you want in the list. Here we simply chose 'Total Gas' and 'n-PrOH'
- Now, you are seeing NaN in some of the entries. 'NaN' is python's way of telling you that, that value was not present in that cell of the excel sheet.
- This means that for the first entry in the experimental results present in the excel file, experimentalist did not record the normality of PrOH.
- There are two most common ways we can handle such cases:

```
1. Replace all such NaN values using zeros using:
   df = df.fillna(0)
   print(df)
2. Replacing all the NaNs with the average value of that property:
   df = df.fillna(df.mean())
   print(df)
```

```
# Showing one of the approaches shown above
# df = df.fillna(0)

# Print contents before we process NaNs
print(df[['Total Gas','n-PrOH']])

# Process the NaNs; replacing them with mean values
df = df.fillna(df.mean())

# Print the newly processed dataframe
print(df[['Total Gas','n-PrOH']])
```

- Also, the dataframe has a column which lists Date of the experiment.
- It does not make sense to see how time of the experiment influences the experiment being conducted because we don't expect Laws of Physics to be time variant!
- So, we will drop that column. We can drop columns in dataframes as,

```
df = df.drop(columns=['DATE'])
```

```
# Dropping irrelevant columns in Pandas dataframe

# df.columns.values lists names of all the columns of the dataframe
# Then we check if the string DATE is present in there or not.
# This returns True if it is present; else returns false
print('Is DATE column present in our dataframe? Ans: ','DATE' in df.columns.values)

# Let's drop the column
df = df.drop(columns=['DATE'])

# Let's check if our command worked!
print('Is DATE column present in our dataframe? Ans: ','DATE' in df.columns.values)
```

- Now, let's look at the column 'Catalyst'

```
In [11]: df['Catalyst']
Out[11]:
0                                                  Cu2O - 2um
1                                                  Cu2O - 2um
2                                                  Cu2O - 2um
3                                                  Cu2O - 2um
4                                                  Cu2O - 2um
                                  ...
3710    Cu(I)Cl Solgel: Cu(I)Cl 40mM + KCl 80mM + H2O ...
3711                          CuF2-derived Cu2O nanosphere
3712                          CuF2-derived Cu2O nanosphere
3713                          CuF2-derived Cu2O nanosphere
3714                          CuF2-derived Cu2O nanosphere
Name: Catalyst, Length: 3715, dtype: object
```

- Since Machine Learning requires all the inputs to be numerical, we want to convert it to numerical values. Some variables however aren't numerical in our present case. For example, type of catalyst in the column 'Catalyst' is not numerical. Such variables are called **Categorical Variables**!
- So, we perform encoding of such categoical something called One Hot Encoding. Pandas has a beautiful function which takes care of it for us!

```
# One-Hot encode your dataframe for categorical variables / features
df = pd.get_dummies(df)
print(df)
```

```
      Nb  Cr  Mo  ...  Electrolyte 2_K HCO3  Electrolyte 2_K OH  Electrolyte 2_Nan
0      0   0   0  ...                     0                   0                 1
1      0   0   0  ...                     0                   0                 1
2      0   0   0  ...                     0                   0                 1
3      0   0   0  ...                     0                   0                 1
4      0   0   0  ...                     0                   0                 1
...   ..  ..  ..  ...                   ...                 ...               ...
3710   0   0   0  ...                     0                   0                 1
3711   0   0   0  ...                     0                   0                 1
3712   0   0   0  ...                     0                   0                 1
3713   0   0   0  ...                     0                   0                 1
3714   0   0   0  ...                     0                   0                 1

[3715 rows x 283 columns]
```

Extracting data based on conditions

- What if you want to extract data from your dataframe based on some conditions?
- Let's look at an example!
- Find all the records / rows where the values corresponding to *Current Density mA/cm2* column are greater than, say 10

```
# select_idx_indicator is a boolean array
select_idx_indicator = df['Current Density mA/cm2'] > 10.0

# df.index is an array composed of index values
select_idx = df.index[select_idx_indicator]

# loc method allows one to select values based on rows and indices
print(df.loc[select_idx])
print(df.loc[select_idx,'Current Density mA/cm2'])
```

Saving the data

- You can save the data to csv or excel format using Pandas
- The way to do is shown below:

```
df.to_csv('filename.csv')
df.to_excel('filename.csv')
```

```
[ ]  # Let's save our processed dataframe to a csv

     # You can replace the string `filename` with the name of your choice!
     df.to_csv('filename.csv')
     # This saves the file in the directory you are at
```

Thus ends the basic intro to PANDAS! Almost there to make Machines Learn!!!

Final step: Let's learn to generate Training and Testing data!

Generating Training and Testing datasets

- We have the processed dataset. Let's generate training and testing datasets out of it
- We can use sklearn's train_test_split function to generate the training and testing dataset!

```
# Let's get input and target values first
X = df.drop(columns=['target'])
y = df['target']

# Let's split the datasets into training and testing!
train_X, test_X, train_y, test_y = train_test_split(X,y,test_size=0.1, random_state=42)
```

- *test_size* parameter determines what fraction of the total dataset to be set aside for generating test dataset
- *random_state* parameter determines how to split to it. It is the seed to the random number generator that lies in the background of the *train_test_split* function

```
# Let's get input and target values first
X = df.drop(columns=['Total Gas'])
y = df['Total Gas']

from sklearn.model_selection import train_test_split

# Let's split the datasets into training and testing!
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1, random_state=42)
```

It is the time to make make Machines Learn!!

Implementing Your ML Model in python

Overview

- Loading Data
- Setting up Model
- Predicting new data points