Machine Learning for Core Engineering Disciplines

Instructor:
Prof. Ananth Govind Rajan
Department of Chemical Engineering
Indian Institute of Science, Bengaluru
Email: ananthgr@iisc.ac.in
Website: https://agrgroup.org

1 Introduction to Python

- Python (https://www.python.org/) is a high-level programming language first introduced in the year 1991, that has increasingly become popular for scientific computing and machine learning applications.

- Python has several packages that can be used to augment its capabilities beyond traditional programming, e.g., to enable graphing and plotting (Matplotlib), scientific computing (SciPy), and linear algebra (NumPy).

- Since it is difficult to install and keep track of several such packages, one typically works with a Python distribution manager, such as Anaconda distribution (https://www.anaconda.com/products/distribution).

- The Anaconda installer for Windows and Macintosh installs not only the Python programming language, but also and some associated packages such as NumPy, Pandas, and Matplotlib. The installation procedure for Linux is more complicated and can be accessed at https://docs.anaconda.com/anaconda/install/linux/.

- Upon installing Anaconda, one also obtains access to an interactive development environment (IDE) called Spyder, which can be used to develop and test Python code. Spyder is also available separately at https://www.spyder-ide.org/.

- It also comes with an interface called the Jupyter notebook to write and execute Python code in an interactive manner. One typically uses Spyder for working with standalone Python code (".py" files) and the Jupyter notebook for working with interactive Python notebooks (".ipynb" files).

- You can access Spyder just by typing it in the search box. To access Jupyter, one can use the Anaconda Distribution's graphical user interface (GUI), or alternatively, follow the steps mentioned below:

1. Open the Anaconda terminal

2. Create a Conda environment with any name, e.g., myenv:

```
conda create -n myenv python=3.9
```

3. Activate the created environment

   ```
   conda activate myenv
   ```

4. Install Jupyter notebook

   ```
   conda install jupyter
   ```

5. Open a Jupyter notebook by typing

   ```
   jupyter notebook
   ```

The Jupyter notebook will be opened in the default web browser, or alternatively, you can copy-paste the link in your browser.

If one decides not to work with Anaconda and/or Spyder (not recommended), one can still install Python and the associated packages manually on a Linux or Macintosh machine. To do so, you can use the following command on the "terminal" of a Linux machine:

```
pip install python3
```

If you use a Macintosh, you will first have to install a package manager call Homebrew (https://brew.sh/), which can be installed using the following command in the "terminal" of a Macintosh:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.
sh)"
```

Note that Homebrew can also be installed on a Linux machine using the same command. Now, the following command can be run using Homebrew on the terminal to install Python:

```
brew install python3
```

After installing Python, Jupyter can be installed on a Linux machine by running the following command on the terminal:

```
pip3 install jupyter
```

On a Macintosh, the following alternative command can be used:

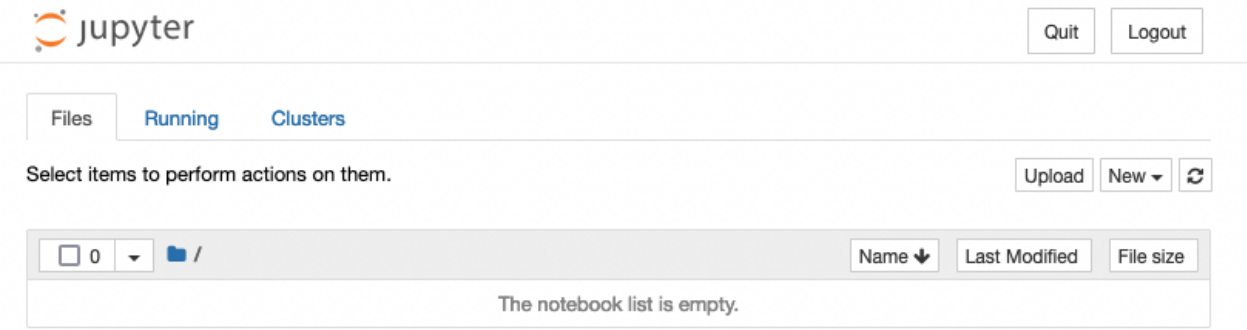
```
brew install jupyter
```

1.1 Some simple commands

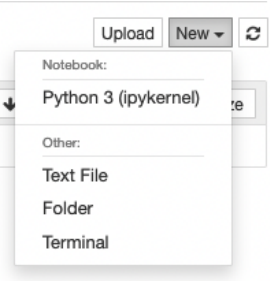
Jupyter uses a web browser to display its programming interface. Recall that, to access Jupyter's Python programming environment, you can use the command:

```
jupyter notebook
```

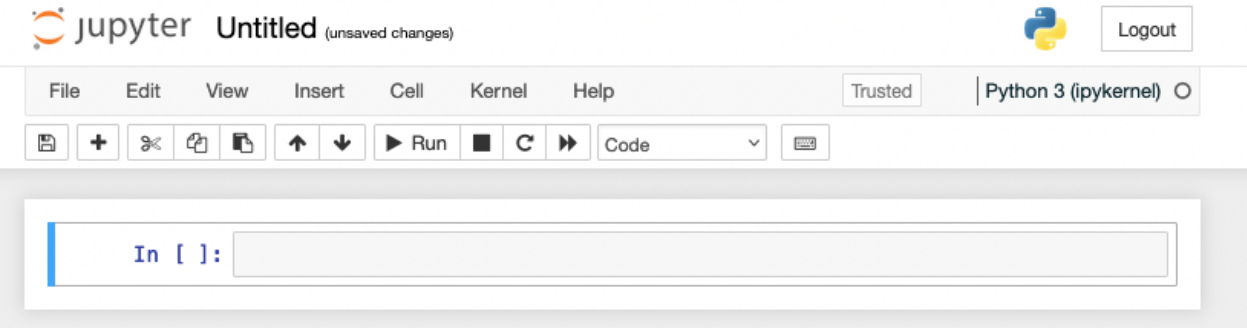
in your regular or Anaconda terminal. This will open up your web browser, which may look as follows:



Here, you can create a new notebook to try out Python commands, clicking on New > Python3 (ipykernel), as shown below:



Once you click on Python3 (ipykernel), a new Jupyter notebook will be created, with your initial screen looking as follows:



Note that the name of the notebook by default is "Untitled". One can click on it to change the name as per their preference.

In Jupyter, each Python command is written in a cell and can be executed independently. We can define a variable and print it as follows:

```
In [1]: a=1

In [2]: print(a)
        1
```

The keyboard shortcut to execute a cell is Shift+Enter. This action can also be performed by selecting Cell > Run Cells from the menu bar. Notice that each cell is assigned a number by Python. To delete a cell, one can choose Edit > Delete Cells from the menu bar or use the keyboard shortcut D,D, which implies that you press the "D" button twice on the keyboard.

Some useful keyboard shortcuts can be found below: (Mac users should replace Ctrl by Cmd.)

- Ctrl + Enter (run the current cell and stay in the same cell)
- Shift + Enter (run the current cell and move to the next cell)

While in the command mode (press Esc to activate and confirm using a blue boundary around the cell), the following commands can be useful:

- Enter (takes you to edit mode)
- A (add a cell above the current cell)
- B (add a cell below the current cell)
- D, D (press D twice to delete the current cell)
- Z (undo cell deletion)
- L (show the line numbers in the cell)

While in the edit mode (indicated by a green boundary around the cell), the following commands can be useful:

- Ctrl + / (commenting/uncommenting the selected lines)
- Tab (indent the selected lines by one tab character)
- Shift + Tab (unindent the selected lines by one tab character)

Apart from integer values, variables can also be used to store text as well as Boolean data. See, e.g., the commands below:

```
In [3]: b=False

In [4]: c="Machine learning"

In [5]: print(b)
        False

In [6]: d=" (ML) "

In [7]: print(c+d)
        Machine learning (ML)
```

The type of a variable can be obtained using the function `type()` as follows:

```
In [8]: type(a)
Out[8]: int

In [9]: type(b)
Out[9]: bool

In [10]: type(c)
Out[10]: str
```

Other typical programming constructs, such as if… elif … else and for loops are also available in Python.

However, note that indentation is very important in Python, and special attention must be given to not mix indentations made using tabs and spaces, which can lead to difficult-to-diagnose errors.

1.2 Scientific computing packages

We will use the NumPy library for helping us carry out numerical operations on matrices and arrays, similar to what can be achieved using the commercial programming language MATLAB.

Another useful Python library is called SciPy and (i) offers various algorithms related to linear algebra, optimization and integration of functions, interpolation of data; (ii) allows the use of special functions, such as exponential, logarithmic, and trigonometric functions in Python; and (iii) enables advanced operations such as fast Fourier transforms, signal and image processing, and the solution of ordinary differential equations.

Finally, Matplotlib is a mathematical graphing and visualization library that allows one to create 2D and 3D plots of data processed in Python. We now discuss the installation of these packages one-by-one.

Note that, when one installs Python via Anaconda or Spyder, most of these packages are already available. Nevertheless, if one would like to separately install NumPy, simply use one of the following commands on your terminal, depending on whether you are using Linux or Macintosh:

```
pip3 install numpy
```

or,

```
brew install numpy
```

Once you do this, you will be able to load the NumPy module in Python using the following command in your Jupyter notebook:

```
import numpy
```

A small modification to the above command can be used to create a shorthand representation for numpy:

```
import numpy as np
```

This will allow us to refer to numpy as np in our Python code, thus making us more efficient in writing code. In a similar manner, you can install SciPy and Matplotlib on your machine and load these into your Jupyter notebook.

1.3 Handling matrices in Python

Before learning to deal with matrices, we will discuss how to handle vectors in Python. The code snippet below uses NumPy to define two vectors in Python and carries out various operations such as addition, subtraction, and multiplication by a scalar, as well as using NumPy to determine the dot and cross product of the two vectors:

```
In [1]: import numpy as np

In [2]: a=np.array([1,2,3])

In [3]: b=np.array([4,5,6])

In [4]: print(a)
        [1 2 3]

In [5]: print(b)
        [4 5 6]

In [6]: a+b
Out[6]: array([5, 7, 9])

In [7]: a-b
Out[7]: array([-3, -3, -3])

In [8]: np.dot(a,b)
Out[8]: 32

In [9]: np.cross(a,b)
Out[9]: array([-3,  6, -3])
```

You should try and explore the various commands that are available in NumPy to handle arrays. We will now proceed to use NumPy to define a matrix in Python and to perform various operations on it, such as calculating its inverse, multiplying the inverse with the original matrix, calculating the eigenvalues and eigenvectors of the matrix, and calculating the determinant of the matrix:

```
In [22]: A=np.array([[-2,1,1],[0,2,1],[1,2,2]])

In [24]: B=np.linalg.inv(A)

In [27]: print(B)
         [[-0.4 -0.   0.2]
          [-0.2  1.  -0.4]
          [ 0.4 -1.   0.8]]

In [34]: I=np.matmul(A,B)

In [35]: print(I)
         [[1. 0. 0.]
          [0. 1. 0.]
          [0. 0. 1.]]

In [46]: np.linalg.eig(A)
Out[46]: (array([-2.20440155,  3.56884961,  0.63555194]),
          array([[ 0.96403988, -0.24065732,  0.08147127],
                 [ 0.06149393, -0.52170607, -0.58916971],
                 [-0.25854518, -0.81847836,  0.80389146]]))

In [47]: np.linalg.det(A)
Out[47]: -5.000000000000001
```

More commands are available within the `linalg` library of NumPy, which you can explore.

Other useful features include the `matrix` library of numpy where you can access the commands `numpy.matrix.transpose`, `numpy.matrix.H`, `numpy.matrix.trace`, etc.

1.4 Handling data in Python

One often stores tabular data in comma separated value (CSV) files. These files can be opened by Microsoft Excel.

Within Python, one can use the Pandas package to read CSV (.csv) and Excel (.xls/.xlsx) files. See, e.g., the below commands to read an Excel file named "Physical_Feature_Data.xlsx" containing data regarding the physical features for 20840 nanopore shapes in graphene, a two-dimensional (2D) material, from the following research article:

Sheshanarayana, R.; Govind Rajan, A. Tailoring Nanoporous Graphene via Machine Learning: Predicting Probabilities and Formation Times of Arbitrary Nanopore Shapes. *J. Chem. Phys.* **2022**, *156*, 204703.

The Excel file is available at:

http://bit.ly/4ks8A7F

or, alternatively, at

https://github.com/agrgroup/MLforNanopores/blob/main/generate_features/data.xlsx

To download the file from the GitHub repository that opens in your web browser, you may have to click on the three dots (…) button on the top right, and then click on "Download".

If you would like to, I encourage to play with any dataset of your choice, pertaining to your own engineering discipline.

Note that the `read_excel` command may not work out of the box, and one may need to use one of the following commands to install the OpenPyXL package:

```
conda install openpyxl
```

or

```
pip install openpyxl
```

```
In [1]: import pandas as pd
```

```
In [5]: pd.read_excel('/Users/ananthgr/Documents/Acads/IISc/Courses/CH 251/Physical_Feature_Data.xlsx')
```

Out[5]:

	num_removed_atoms	num_ZZ_atoms	num_AC_atoms	num_UA_atoms	num_SB_atoms	count_rim_atoms	num_incomplete_hexagons	num_5_membered
0	4	6	0	0	0	18	6	
1	4	4	2	0	0	18	6	
2	4	6	0	0	0	18	6	
3	5	5	2	0	0	20	7	
4	5	5	0	0	1	19	7	
...	...	...	...	...	...	...	...	
20835	22	8	8	2	0	42	21	
20836	22	7	6	1	1	37	20	
20837	22	6	8	0	1	37	20	
20838	22	8	8	0	0	38	20	
20839	22	10	4	0	1	37	20	

20840 rows × 24 columns

The data in the Excel file can also be stored in a data structure, as follows:

```
In [6]: data=pd.read_excel('/Users/ananthgr/Documents/Acads/IISc/Courses/CH 251/Physical_Feature_Data.xlsx')
```

The type of the data structure can be seen as follows:

```
In [7]: type(data)
Out[7]: pandas.core.frame.DataFrame
```

Finally, one can determine the number of rows (data points) and columns (features) in the dataset as follows:

```
In [8]: data.shape
Out[8]: (20840, 24)
```

Note that pressing tab after writing the name of a variable/package shows a list of commands that can be executed.

One can see that there are 20,840 data points in the dataset. Furthermore, there are 24 columns in the data set.

The first 22 columns contain the features of the graphene nanopores and the last 2 columns contain, respectively, the formation time and probability of formation of each nanopore.

Thus, there are 2 target variables and 22 features in our dataset. One could access some rows from either the beginning or the end of the dataset as follows:

```
In [14]: data[3:6]
```

Out[14]:

	num_removed_atoms	num_ZZ_atoms	num_AC_atoms	num_UA_atoms	num_SB_atoms	count_rim_atoms	num_incomplete_hexagons	num_5_membered_ring
3	5	5	2	0	0	20	7	
4	5	5	0	0	1	19	7	
5	5	7	0	0	0	20	7	

3 rows × 24 columns

```
In [15]: data[-3:]
```

Out[15]:

	num_removed_atoms	num_ZZ_atoms	num_AC_atoms	num_UA_atoms	num_SB_atoms	count_rim_atoms	num_incomplete_hexagons	num_5_membered
20837	22	6	8	0	1	37	20	
20838	22	8	8	0	0	38	20	
20839	22	10	4	0	1	37	20	

3 rows × 24 columns

One can also sort the values as per the data in a certain column and only display certain columns in the dataset, e.g., using the command shown below:

```
In [24]: data[['num_removed_atoms','probability']].sort_values('probability',ascending=False)[0:5]
```

Out[24]:

	num_removed_atoms	probability
3	5	0.4709
1	4	0.4037
0	4	0.3956
20	7	0.3840
6	5	0.2553

It is also possible to determine the maximum, minimum, and mean value of a certain feature, as follows:

```
In [27]: data[['probability']].max()
Out[27]: probability    0.4709
         dtype: float64

In [28]: data[['probability']].min()
Out[28]: probability    0.0001
         dtype: float64

In [29]: data[['probability']].mean()
Out[29]: probability    0.000912
         dtype: float64
```

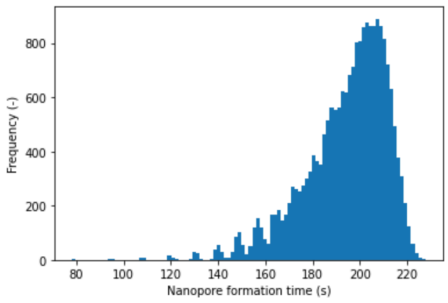
1.5 Plotting data in Python

Plotting data in Python is made simple by the use of the Matplotlib package. For this purpose, it is useful to load the Pyplot module from Matplotlib.

```
In [44]: import matplotlib.pyplot as plt
```

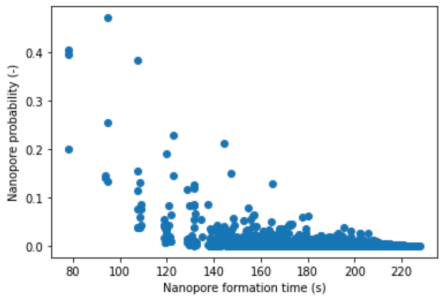
Subsequently, several types of plots can be made. Our first example is that of a bar plot:

```
In [49]: plt.hist(data['time'],bins=100)
         plt.xlabel('Nanopore formation time (s)')
         plt.ylabel('Frequency (-)')
Out[49]: Text(0, 0.5, 'Frequency (-)')
```



Another example is that of a scatter plot:

```
In [53]: plt.scatter(data['time'],data['probability'])
         plt.xlabel('Nanopore formation time (s)')
         plt.ylabel('Nanopore probability (-)')
Out[53]: Text(0, 0.5, 'Nanopore probability (-)')
```



1.6 Some Python-based ML packages: scikit-learn and PyTorch

scikit-learn, also known as sklearn, is a Python library that offers several ML algorithms for classification and regression learning, as well as for unsupervised learning.

On the other hand, PyTorch is a Python library for implementing neural networks and deep learning algorithms.

The installation procedure for scikit-learn is very simple. You can use the following command in the terminal on both Linux and Macintosh machines:

```
pip3 install -U scikit-learn
```

More instructions can be found at https://scikit-learn.org/stable/install.html.

This module can be loaded in Python using the command:

```
import sklearn
```

Next, we will install the PyTorch module in Python to work with neural networks. You can install PyTorch using the command below:

```
pip3 install torch
```

More information can be found here: https://pytorch.org/get-started/locally/.

Sometimes, although you may have installed the scikit-learn and PyTorch modules, they may not be getting imported in Jupyter, with the notebook displaying an error "ModuleNotFoundError: No module named …". This may happen if the Python installed on your system and in the Jupyter notebook have different versions. To check the Python version, use the following command on your terminal:

```
python --version
```

On the other hand, the Python version on Jupyter can be checked using:

```
from platform import python_version
python_version()
```

If these versions are not the same, you can use the following commands on your terminal:

```
pip3 install ipykernel --upgrade
python3 -m ipykernel install –user
```

After running these commands and restarting your Jupyter notebook, you should be able to import the scikit-learn and PyTorch packages into Python via the Jupyter notebook.