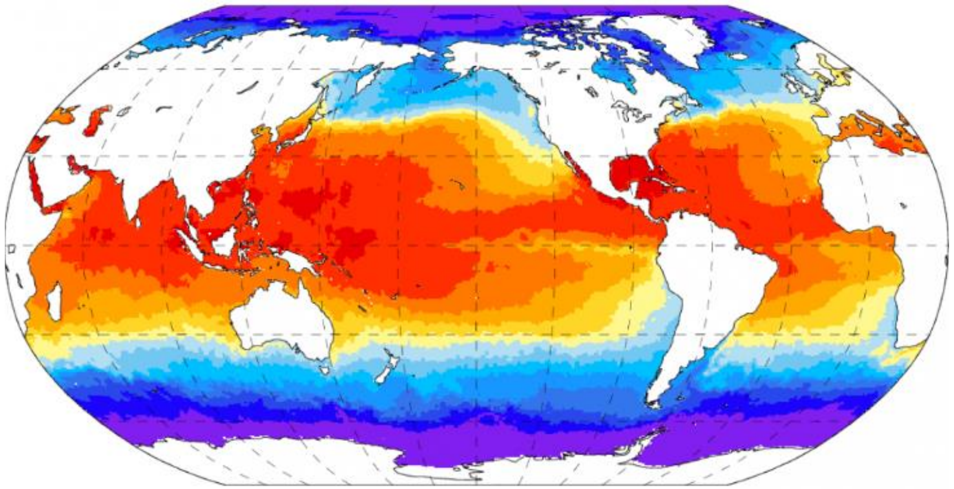
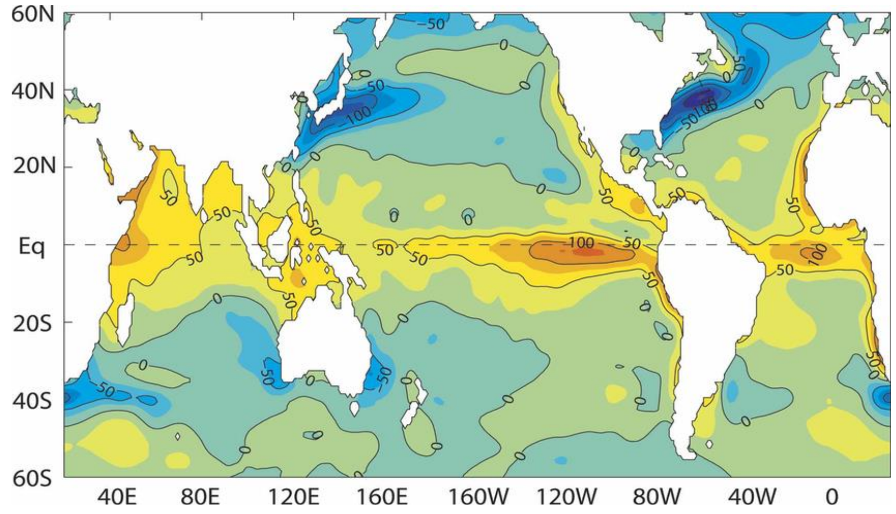
Summer Research Fellowship

SFRP–2020

Under TEQUIP - III
August 2020

Python For Plotting Meteorological Data





Pranav Bahl
Undergraduate Student
Delhi Technological University
(Formerly Delhi College of Engineering)

Dr. Vamsi K Chalamalla
Assistant Professor
Department of Applied Mechanics
Indian Institute of Technology Delhi

Installing Packages

- **First step**: Open Anaconda Command Prompt in your Windows/Mac.
- **Second step**: Install/Update all necessary packages/modules using following commands.

```
conda update pip

conda update numpy


conda update matplotlib

conda install netcdf4

conda install basemap
```

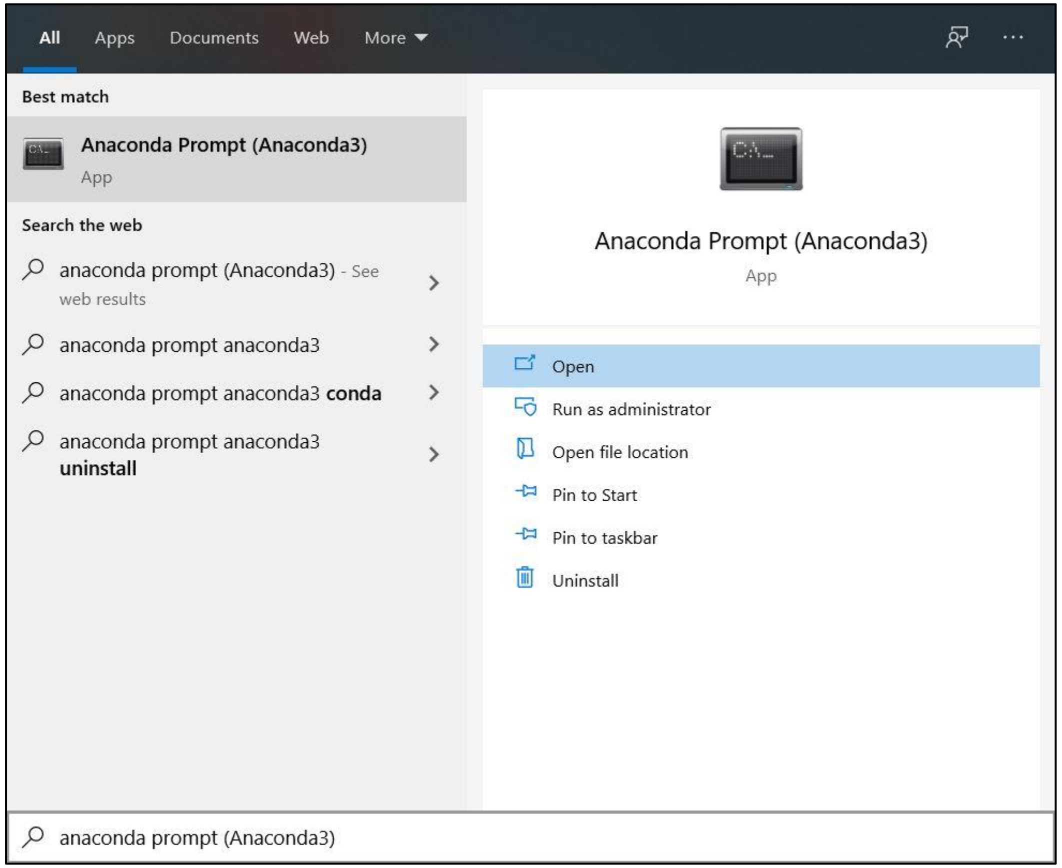
- **Note:** Use "conda install" for modules not present instead of "conda update", before updating .

Anaconda Command Prompt

1) Go to the search section in your windows and type “Anaconda prompt”.

2) Click on the prompt icon (as shown on the right side).

3) Once you click on the icon, you will be directed to the Anaconda terminal from where you can install packages and can access jupyter notebook.



Installing Numpy

Enter the command “conda install numpy”, in your Anaconda prompt terminal

```
Anaconda Prompt (Anaconda3)
(base) C:\Users\purub> conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\purub\Anaconda3

  added / updated specs:
    - numpy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    mkl_fft-1.1.0              |   py37h45dec08_0         116 KB
    mkl_random-1.1.1           |   py37h47e9c7a_0         233 KB
    numpy-1.19.1               |   py37h5510c5b_0          22 KB
    numpy-base-1.19.1          |   py37ha3acd2a_0         3.8 MB
    ------------------------------------------------------------
                                           Total:         4.2 MB
```

Enter “y” in the “Proceed ([y]/n)?” section, once all the new packages to be installed are listed.

```
The following NEW packages will be INSTALLED:

  blas               pkgs/main/win-64::blas-1.0-mkl
  intel-openmp       pkgs/main/win-64::intel-openmp-2020.1-216
  mkl                pkgs/main/win-64::mkl-2020.1-216
  mkl-service        pkgs/main/win-64::mkl-service-2.3.0-py37hb782905_0
  mkl_fft            pkgs/main/win-64::mkl_fft-1.1.0-py37h45dec08_0
  mkl_random         pkgs/main/win-64::mkl_random-1.1.1-py37h47e9c7a_0
  numpy              pkgs/main/win-64::numpy-1.19.1-py37h5510c5b_0
  numpy-base         pkgs/main/win-64::numpy-base-1.19.1-py37ha3acd2a_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
numpy-base-1.19.1    | 3.8 MB    | ############################################################################ | 100%
mkl_fft-1.1.0        | 116 KB    | ############################################################################ | 100%
mkl_random-1.1.1     | 233 KB    | ############################################################################ | 100%
numpy-1.19.1         | 22 KB     | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Installing netCDF4

Enter the command “conda install netcdf4”, in your Anaconda prompt terminal

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\purub> conda install netcdf4
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\purub\Anaconda3

  added / updated specs:
    - netcdf4


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    curl-7.71.1                |       h2a8f88b_1         129 KB
    krb5-1.18.2                |       hc04afaa_0         702 KB
    libcurl-7.71.1             |       h2a8f88b_1         275 KB
    libnetcdf-4.7.3            |       h1302dcc_0         516 KB
    libssh2-1.9.0              |       h7a1dbc1_1         215 KB
    netcdf4-1.5.3              |   py37h012c1a0_0         353 KB
    ------------------------------------------------------------
                                           Total:         2.1 MB
```

Enter “y” in the “Proceed ([y]/n)?” section, once all the new packages to be installed are listed.

```
The following NEW packages will be INSTALLED:

  cftime             pkgs/main/win-64::cftime-1.2.1-py37h2a96729_0
  curl               pkgs/main/win-64::curl-7.71.1-h2a8f88b_1
  hdf4               pkgs/main/win-64::hdf4-4.2.13-h712560f_2
  hdf5               pkgs/main/win-64::hdf5-1.10.4-h7ebc959_0
  icc_rt             pkgs/main/win-64::icc_rt-2019.0.0-h0cc432a_1
  krb5               pkgs/main/win-64::krb5-1.18.2-hc04afaa_0
  libcurl            pkgs/main/win-64::libcurl-7.71.1-h2a8f88b_1
  libnetcdf          pkgs/main/win-64::libnetcdf-4.7.3-h1302dcc_0
  libssh2            pkgs/main/win-64::libssh2-1.9.0-h7a1dbc1_1
  netcdf4            pkgs/main/win-64::netcdf4-1.5.3-py37h012c1a0_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
netcdf4-1.5.3        | 353 KB    | ############################################################################ | 100%
libcurl-7.71.1       | 275 KB    | ############################################################################ | 100%
libnetcdf-4.7.3      | 516 KB    | ############################################################################ | 100%
libssh2-1.9.0        | 215 KB    | ############################################################################ | 100%
curl-7.71.1          | 129 KB    | ############################################################################ | 100%
krb5-1.18.2          | 702 KB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Installing matplotlib and basemap

Enter the command “conda install matplotlib”, in your Anaconda prompt terminal

Enter “y” in the “Proceed ([y]/n)?” section, once all the new packages to be installed are listed.

Anaconda Prompt (Anaconda3) - python

```
(base) C:\Users\purub>conda install matplotlib
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\purub\Anaconda3

  added / updated specs:
    - matplotlib


The following NEW packages will be INSTALLED:

  cycler             pkgs/main/win-64::cycler-0.10.0-py37_0
  kiwisolver         pkgs/main/win-64::kiwisolver-1.2.0-py37h74a9793_0
  matplotlib         pkgs/main/win-64::matplotlib-3.2.2-0
  matplotlib-base    pkgs/main/win-64::matplotlib-base-3.2.2-py37h64f37c6_0


Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Enter the command “conda install basemap”, in your Anaconda prompt terminal

Enter “y” in the “Proceed ([y]/n)?” section, once all the new packages to be installed are listed.

Anaconda Prompt (Anaconda3) - python

```
(base) C:\Users\purub>conda install basemap
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\purub\Anaconda3

  added / updated specs:
    - basemap


The following NEW packages will be INSTALLED:

  basemap            pkgs/main/win-64::basemap-1.2.0-py37h4e5d7af_0
  geos               pkgs/main/win-64::geos-3.6.2-h9ef7328_2
  proj4              pkgs/main/win-64::proj4-5.2.0-ha925a31_1
  pyproj             pkgs/main/win-64::pyproj-1.9.6-py37h6782396_0
  pyshp              pkgs/main/noarch::pyshp-2.1.0-py_0


Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Checking Modules/Packages

Once all the required modules/packages are installed, you could run following commands in terminal and check if they are installed properly.

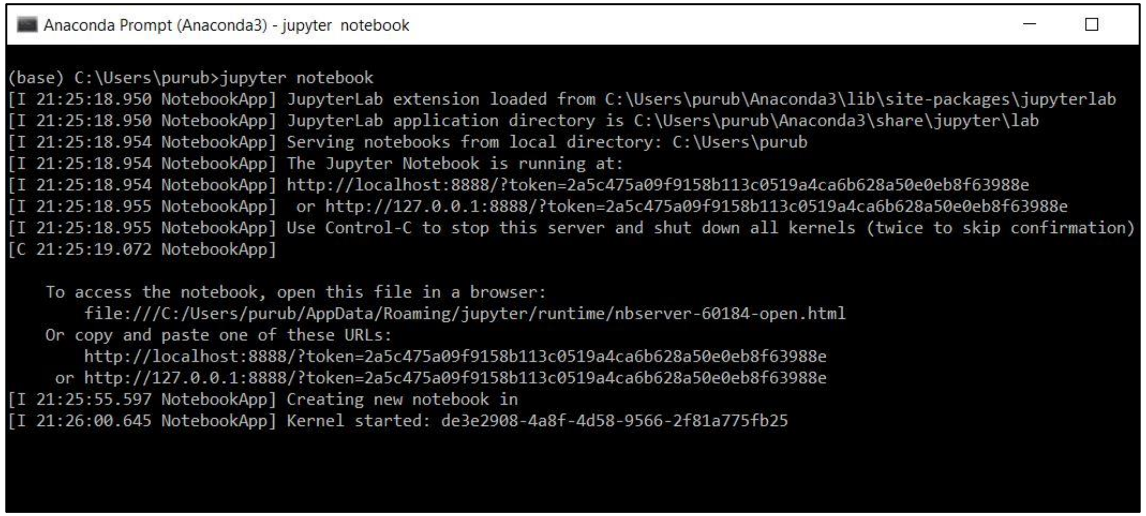
1)Enter command “python”, it will allow you to execute python code.
2)Enter following commands as shown in the screenshot one by one and press “Enter”, if they do not give back any errors then they are installed.
3)Enter “Ctrl + Z” or “exit()” to come back at Anaconda prompt terminal

Anaconda Prompt (Anaconda3) - python

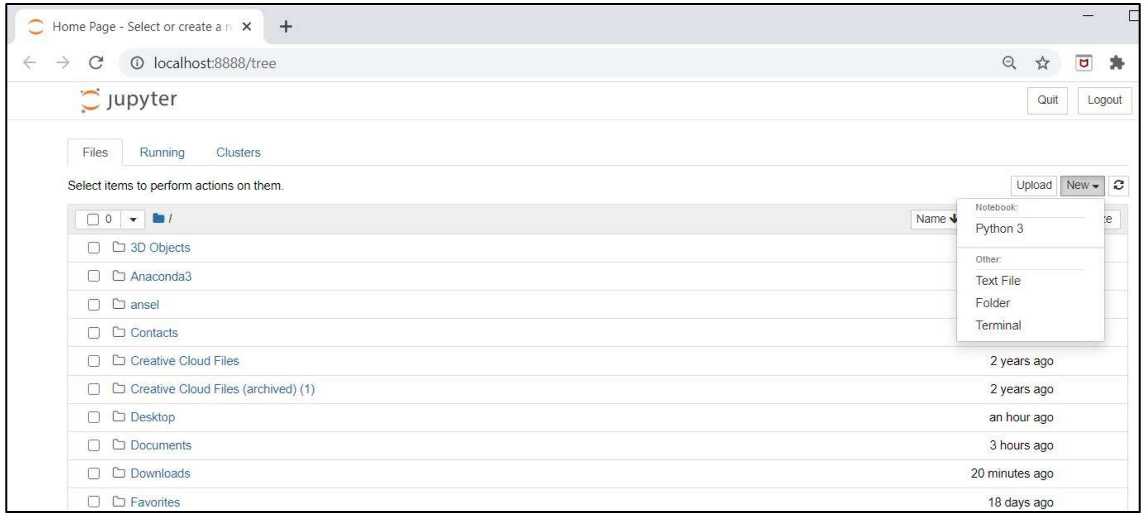
```
(base) C:\Users\purub>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from mpl_toolkits.basemap import Basemap
>>> from matplotlib import pyplot
>>> import numpy as np
>>> from netCDF4 import Dataset
>>>
```

Jupyter Notebook

Enter the command “jupyter notebook”, in your Anaconda prompt terminal to access jupyter notebook. You will be automatically directed to your web browser.



Once you are in the notebook you can select the path from where you want to access/ save your jupyter notebook file.



Numpy

- Open-source library (module) for Python.
- Support for large multi-dimensional arrays.
- Offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms etc.
- The core of NumPy is a well-optimized C code. Provides speed of a compiled code.
- High level syntax makes it easy for use.

More information and documentation can be found on the website

NumPy

https://numpy.org/

Numpy (Basics)

Here we generate values from 0 to 3 using “np.arrange()” function of numpy and reshape it into 2 dimensional array/matrix using “reshape” function.

```
import numpy as np
a = np.arange(4)
print(a)
a = np.arange(4).reshape(2,2)
print(a)
```

```
[0 1 2 3]
[[0 1]
 [2 3]]
```

Here we generate matrices containing either only zeroes or only ones using numpy functions “np.zeros()” and “np.ones()”. The shape of the matrices are fed in the format (n,m), where “n” are the rows and “m” are the columns

```
np.zeros((3, 4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
np.ones((3, 4))
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Values from 0 to 7 are generated using “np.arrange()” function and then reshaped into 1 dimensional, 2 dimensional and 3 dimensional array/matrix using the “.reshape()” function.

```
a = np.arange(8)   # 1d array
print(a)
```

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

```
a = np.arange(8).reshape(2,4)   # 2d array
print(a)
```

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

```
a = np.arange(8).reshape(2,2,2)   # 3d array
print(a)
```

```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

```
a.shape
```

```
(2, 2)
```

```
a = np.array([1,2,3,4])
print(a)
```

```
[1 2 3 4]
```

Here we check the shape of array/matrix containing “n” rows and “m” columns using “shape” function in the form of (n,m). Also an array can be generated using the function “np.array()” available in numpy

Numpy (Basics)

Here we generated two arrays “a” and “b” ,then subtraction operation was performed on them element wise. Also element wise array “b” has been squared. Similarly other unary operations can also be performed.

```
a = np.array( [4,5,6] )
b = np.arange( 3 )
c = a-b
print(c)
```

```
[4 4 4]
```

```
b**2
```

```
array([0, 1, 4], dtype=int32)
```

```
np.sin(b)
```

```
array([0.        , 0.84147098, 0.90929743])
```

```
a<5
```

```
array([ True, False, False])
```

Element wise “sine” function was performed on the “b” array. Also element wise a Boolean array has been generated for array “a” with “a < 5”.

Here we have performed elementwise product and matrix product for generated 2*2 matrices. Also various operations has been performed on array “a” as shown below.

```
A = np.array( [[1,1], [0,1]] )
B = np.array( [[2,0],[3,4]] )

c = A*B  # elementwise product
print(c)

c = A@B  # matrix product
print(c)
```

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
```

```
a = np.arange(4)
print(a)
print("sum =",a.sum())
print("min =",a.min())
print("max =",a.max())
print("exp =",np.exp(a))
print("sqrt =",np.sqrt(a))
```

```
[0 1 2 3]
sum = 6
min = 0
max = 3
exp = [ 1.          2.71828183  7.3890561  20.08553692]
sqrt = [0.         1.         1.41421356 1.73205081]
```

```
a = np.arange(4).reshape(2,2)
print("sum =",a.sum(axis=1))
print("sum =",a.min(axis=0))
```

```
sum = [1 5]
sum = [0 1]
```

```
a = np.arange(5)**3
print(a)
print(a[2])
print(a[2:5])
```

```
[ 0  1  8 27 64]
8
[ 8 27 64]
```

In the first part, “row wise” sum has been performed on 2*2 matrix and also “column wise” minimum values are generated for the same . In the second part, slicing operation has been performed on array “a” for extracting particular values using index values.

Numpy (Advantages)

For numpy no "for loops" are needed and hence it is very efficient and fast. In the example shown below the time taken by numpy for the same operation was 15.5 times faster.

```
import time
import numpy

X = list(range(10000000))
Y = list(range(10000000))

t1 = time.time()

Z = [0]*10000000
for i in range(10000000):
    Z[i] = X[i] + Y[i]

print("time taken by traditional method :",time.time() - t1)


X = numpy.arange(10000000)
Y = numpy.arange(10000000)
t1 = time.time()

Z = X + Y

print("time taken by numpy :",time.time() - t1)
```

```
time taken by traditional method : 2.008618116378784
time taken by numpy : 0.1304466724395752
```

In both of these examples, simple matrix manipulations were done using much fewer lines of code and in a much efficient and faster way.

```
import numpy as np

X = np.ones((2,2))
print("numpy output", 2*X)

Y = [[0]*2, [0]*2]
X = [[1]*2, [1]*2]

for i in range(2):
  for j in range(2):
    Y[i][j]=2*X[i][j]

print("Conventional method output",Y)
```

numpy

Lists

```
numpy output [[2. 2.]
 [2. 2.]]
Conventional method output [[2, 2], [2, 2]]
```

```
import numpy as np

A = np.arange(4).reshape(2,2)
B = np.arange(4).reshape(2,2)
K = A+B
print("numpy output (K):",K)
print("numpy output (K_sqr):",K*K)


A = [list(range(2)), list(range(2,4))]
B = [list(range(2)), list(range(2,4))]
K = [[0]*2, [0]*2]

for i in range(2):
  for j in range(2):
    K[i][j]=A[i][j]+B[i][j]
print("Conventional method output (K):",K)

K_sqr = [[0]*2, [0]*2]
for i in range(2):
  for j in range(2):
    K_sqr[i][j]=K[i][j]*K[i][j]
print("Conventional method output (K_sqr):",K_sqr)
```

numpy

Lists

```
numpy output (K): [[0 2]
 [4 6]]
numpy output (K_sqr): [[ 0  4]
 [16 36]]
Conventional method output (K): [[0, 2], [4, 6]]
Conventional method output (K_sqr): [[0, 4], [16, 36]]
```

Matplotlib

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Full control over customization . ex : line styles, font properties, axes properties etc.
- Export and embed plots to a number of file formats.
- High level syntax makes it easy to plot, with just a few lines of code.

More information and documentation can be found on the website

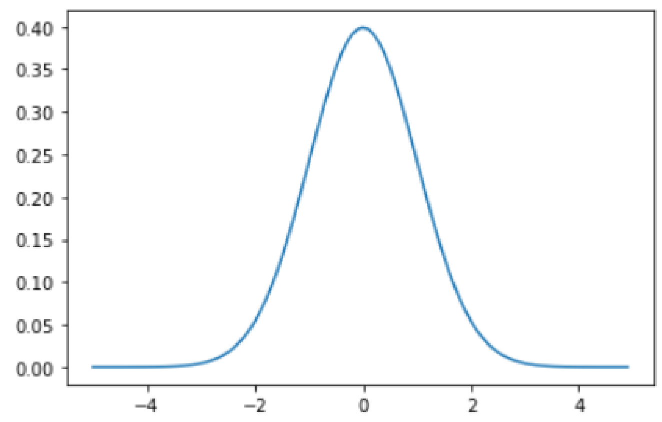
matplotlib
Version 3.3.0

https://numpy.org/

Matplotlib

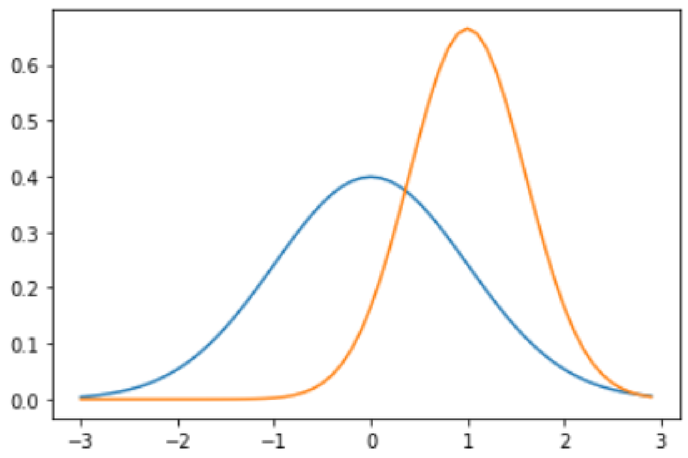
A probability density function has been plotted here using matplotlib, centred at location 0.0 with scale 1.0. This is a Line graph plotting style.

```
x = np.arange(-5, 5, 0.1)

plt.plot(x, norm.pdf(x))
plt.show()
```



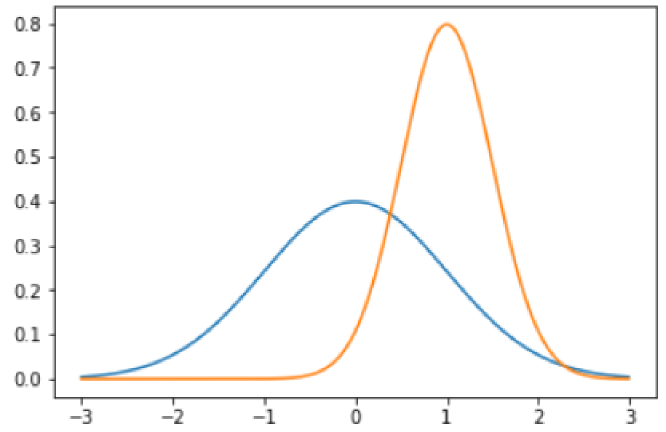
Two probability density function plots have been plotted here simultaneously on a single graph. The other pdf being centred at 1.0 and with scale 0.6

```
plt.plot(x, norm.pdf(x))
plt.plot(x, norm.pdf(x, 1.0, 0.6))
plt.show()
```



Here this plot has been saved in “png” format with the name “MyPlot”, in the path where jupyter notebook is running.

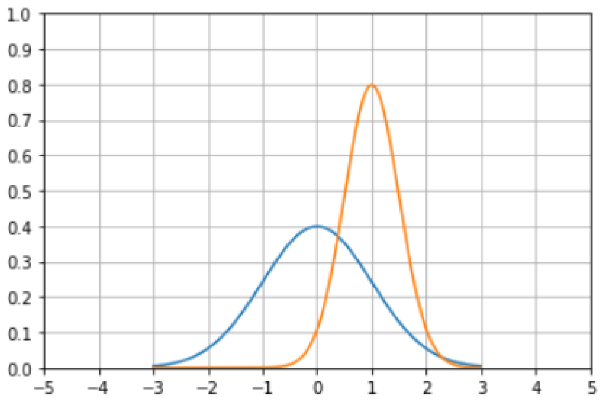
```
plt.plot(x, norm.pdf(x))
plt.plot(x, norm.pdf(x, 1.0, 0.5))
plt.savefig('MyPlot.png', format='png')
```



Matplotlib

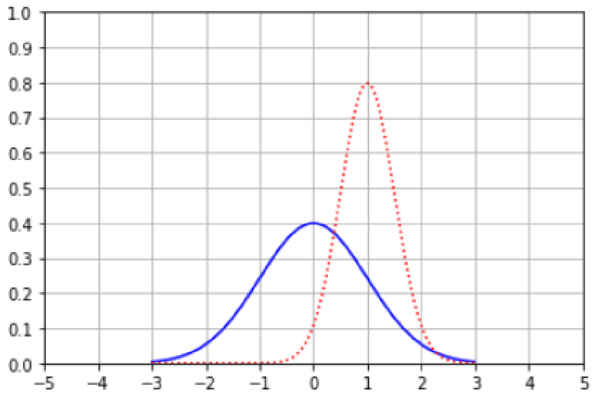
Here the axes of the plot are modified by accessing it through "plt.axes()". "set_xlim"/"set_ylim" are used to set the limit of th plots. Also "set_xticks"/"set_yticks" are used to set the tick values on the axes. Also grids are generated on the plot using "axes.grids()"

```
axes = plt.axes()
axes.set_xlim([-5, 5])
axes.set_ylim([0, 1.0])
axes.set_xticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
axes.set_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
axes.grid()
plt.plot(x, norm.pdf(x))
plt.plot(x, norm.pdf(x, 1.0, 0.5))
plt.show()
```



Here the colours of the pdf plots are modified using different predefined letters for ex: "b", "r" etc. in the "plt.plot()" command. The plots can also be customized to dotted or dashed version using these symbol (**:** **,** **-** etc.) after colour alphabet for ex "**r:**" , "**r-**"

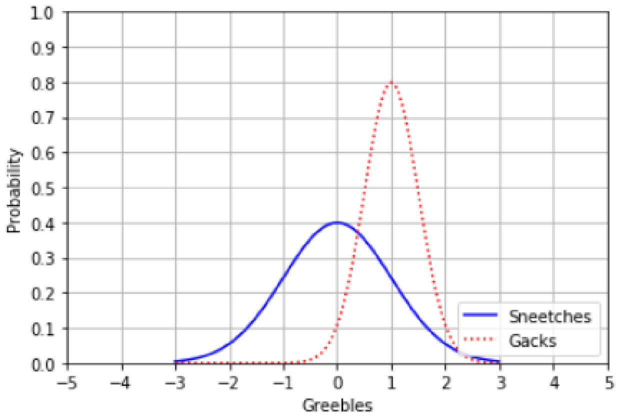
```
axes = plt.axes()
axes.set_xlim([-5, 5])
axes.set_ylim([0, 1.0])
axes.set_xticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
axes.set_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
axes.grid()
plt.plot(x, norm.pdf(x), 'b-')
plt.plot(x, norm.pdf(x, 1.0, 0.5), 'r:')
plt.show()
```



Matplotlib

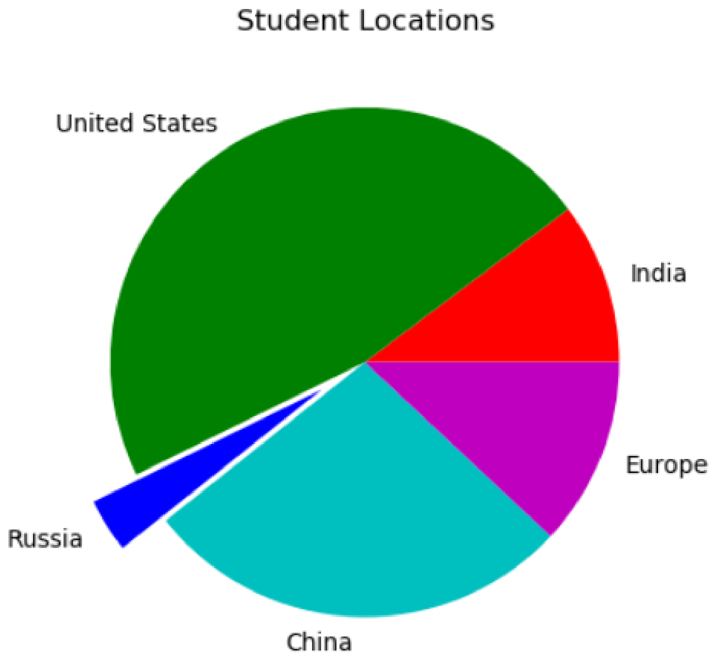
Here the axes have been labelled using function "plt.xlabel()"/"plt.ylabel()" and also Legends have been named using "plt.legend()", "loc" represents out of which 4 corners you need legend box to be on.

```
axes = plt.axes()
axes.set_xlim([-5, 5])
axes.set_ylim([0, 1.0])
axes.set_xticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
axes.set_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
axes.grid()
plt.xlabel('Greebles')
plt.ylabel('Probability')
plt.plot(x, norm.pdf(x), 'b-')
plt.plot(x, norm.pdf(x, 1.0, 0.5), 'r:')
plt.legend(['Sneetches', 'Gacks'], loc=4)
plt.show()
```



A pie chart has been plotted here. Values, colours and labels are fed in the "plt.pie()" function for plotting the pie chart. Explode refers to the percentage of the section to be exploded out of the chart which is to be highlighted.

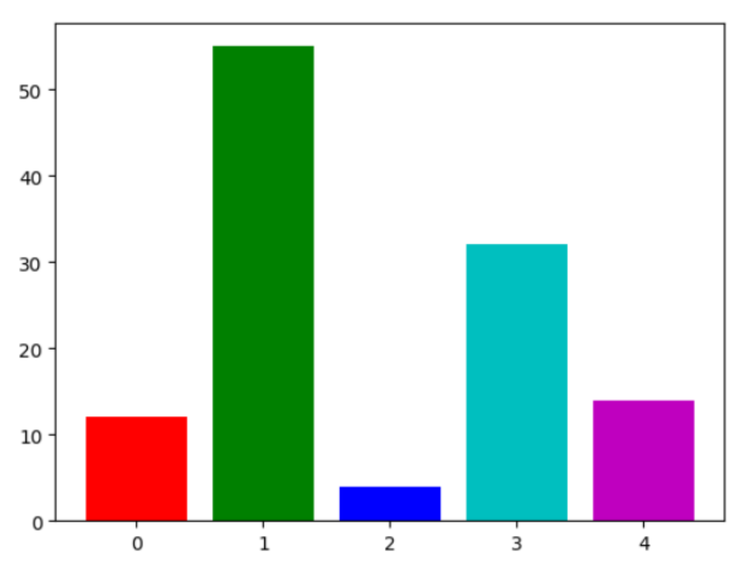
```
values = [12, 55, 4, 32, 14]
colors = ['r', 'g', 'b', 'c', 'm']
explode = [0, 0, 0.2, 0, 0]
labels = ['India', 'United States', 'Russia', 'China', 'Europe']
plt.pie(values, colors= colors, labels=labels, explode = explode)
plt.title('Student Locations')
plt.show()
```



Matplotlib

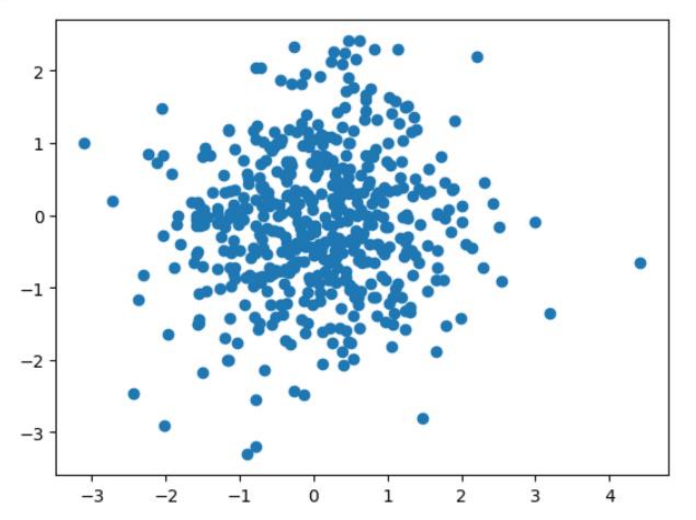
A bar graph has been plotted here with the mentioned values and in the corresponding colours. Range in the x axis has been set from 0 to 4.

```
values = [12, 55, 4, 32, 14]
colors = ['r', 'g', 'b', 'c', 'm']
plt.bar(range(0,5), values, color= colors)
plt.show()
```



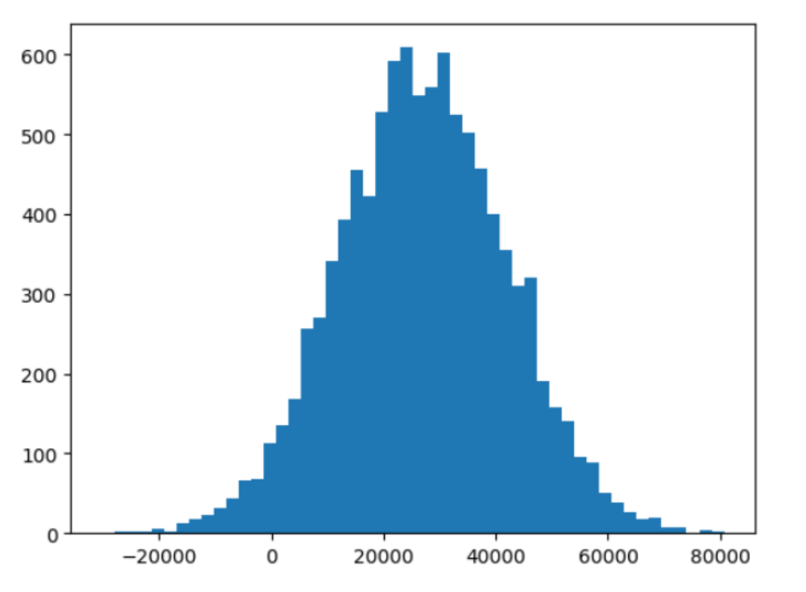
A scatter plot has been plotted here with randomly generated 500 values on the graph.

```
from pylab import randn

X = randn(500)
Y = randn(500)
plt.scatter(X,Y)
plt.show()
```



A histogram has been plotted here. With the centre of distribution being at 27000 with a standard deviation of 15000 and sample size of 10000.

```
incomes = np.random.normal(27000, 15000, 10000)
plt.hist(incomes, 50)
plt.show()
```



Pandas

- Open-source library for Python.
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format
- Flexible reshaping and pivoting of data sets.
- Highly optimized for performance, with critical code paths written in Cython or C.
- High level syntax makes it easy for use.

More information and documentation can be found on the website

pandas

https://pandas.pydata.org/

Pandas

Here we create a Data Frame by passing random values using numpy, setting datetime as index and with labelling of columns. The frequency of dates is in days and can be changed to months or years. To view the data from the beginning or from the bottom following functions could be used ".head(n)"/".tail(n)" where "n" is the number of the data points.

```
import pandas as pd
import numpy as np

dates = pd.date_range('20130101', periods=4)
df = pd.DataFrame(np.random.randn(4, 2), index=dates, columns=list('AB'))

df
```

	A	B
2013-01-01	0.869438	-0.607724
2013-01-02	1.370371	0.637297
2013-01-03	-0.197823	-1.486159
2013-01-04	0.605245	-1.856723

```
print(df.head(2))
print(df.tail(2))
```

```
                   A         B
2013-01-01  0.869438 -0.607724
2013-01-02  1.370371  0.637297
                   A         B
2013-01-03 -0.197823 -1.486159
2013-01-04  0.605245 -1.856723
```

In the first part, using function "df.describe()" we have generated a quick static summary of the data. In the second part with function "df.T" we have taken the transpose of our dataframe.

```
df.describe()
```

	A	B
count	4.000000	4.000000
mean	0.661808	-0.828327
std	0.655066	1.108612
min	-0.197823	-1.856723
25%	0.404478	-1.578800
50%	0.737341	-1.046941
75%	0.994671	-0.296469
max	1.370371	0.637297

```
df.T
```

	2013-01-01	2013-01-02	2013-01-03	2013-01-04
A	0.869438	1.370371	-0.197823	0.605245
B	-0.607724	0.637297	-1.486159	-1.856723

Pandas

In the first part a series corresponding to column “A” has been printed. In the second part the slicing operation has been performed, it is evident that either of the index notations can be used to slice the desired data.

```
df['A']
```

```
2013-01-01    0.869438
2013-01-02    1.370371
2013-01-03   -0.197823
2013-01-04    0.605245
Freq: D, Name: A, dtype: float64
```

```
df[0:3]
```

	A	B
2013-01-01	0.869438	-0.607724
2013-01-02	1.370371	0.637297
2013-01-03	-0.197823	-1.486159

```
df['20130102':'20130104']
```

	A	B
2013-01-02	1.370371	0.637297
2013-01-03	-0.197823	-1.486159
2013-01-04	0.605245	-1.856723

In the first part data corresponding to a datetime index has been printed by with help of the **labels**. In the second/third part they have been sliced with the help of their **postions**. In the fourth part a series has been generated with an automatic indexing using the “date_range”.

```
df.loc['20130102', ['A', 'B']]
```

```
A    1.370371
B    0.637297
Name: 2013-01-02 00:00:00, dtype: float64
```

```
df.iloc[3]
```

```
A    0.605245
B   -1.856723
Name: 2013-01-04 00:00:00, dtype: float64
```

```
df.iloc[0:2, 0:1]
```

	A
2013-01-01	0.869438
2013-01-02	1.370371

```
s1 = pd.Series([1, 2, 3], index=pd.date_range('20130102', periods=3))
s1
```

```
2013-01-02    1
2013-01-03    2
2013-01-04    3
Freq: D, dtype: int64
```

Pandas

In the first case mean of the data has been generated along the column (axis=0 : by default). In the next section the mean has been calculated along the rows (axis =1).

```
df.mean()
```

```
A    0.661808
B   -0.828327
dtype: float64
```

```
df.mean(1)
```

```
2013-01-01    0.130857
2013-01-02    1.003834
2013-01-03   -0.841991
2013-01-04   -0.625739
Freq: D, dtype: float64
```

```
m1 = pd.DataFrame({'A': ["Mango", "Apple"], 'B': [3, 4]})
m2 = pd.DataFrame({'A': ["Mango", "Apple"], 'C': [7, 8]})
m1
```

	A	B
0	Mango	3
1	Apple	4

Two “DataFrame” has been generated here and merged using the command “pd.merge”. A series has been generated with an automatic alignment with the index using the index frequency as months, which can be changed to years by replacing freq = ‘M’ with freq =‘Y’.

```
m2
```

	A	C
0	Mango	7
1	Apple	8

```
pd.merge(m1, m2, on='A')
```

	A	B	C
0	Mango	3	7
1	Apple	4	8

```
Z1 = pd.date_range('3/6/2012', periods=5, freq='M')
T1 = pd.Series(np.arange(5), Z1)
T1
```

```
2012-03-31    0
2012-04-30    1
2012-05-31    2
2012-06-30    3
2012-07-31    4
Freq: M, dtype: int32
```

Pandas

With this command, a csv file is read as a DataFrame and the data stored inside it can be manipulated in the discussed ways.

Note : csv file should be in the same path where you are running your notebook.

```
data = pd.read_csv('SstDataSouth_BayOfBengalMonthly.csv')
```

```
df.to_csv('JanuarySstDataSouth_BayOfBengalMonthly.csv')
```

With this command a DataFrame can be saved in csv format. The file will be downloaded in the path where you are running your notebook. The syntax of the command is as follows "DataFrame.to_csv('NameOfTheFile')".

Using the following commands the data series time span representations are altered. Using the command ".to_period()" the time index is changed to "year-month" and with the command ".to_timestamp()" time index has been changed to "year-month-day".

```
Z1.to_period()

PeriodIndex(['2012-03', '2012-04', '2012-05', '2012-06', '2012-07'], dtype='period[M]', freq='M')

Z2.to_timestamp()

DatetimeIndex(['2012-03-01', '2012-04-01', '2012-05-01', '2012-06-01',
               '2012-07-01'],
              dtype='datetime64[ns]', freq='MS')
```

Google Colaboratory

- Colaboratory allows you to write and execute Python in your browser, with free access to GPUs
- It functions same as Jupyter Notebook, but stored on Google Drive.
- Contains several Third party in-built visualization libraries.
- Files can be easily shared with the help of Google Drive
- Files are stored in standard jupyter notebook format.

More information and documentation can be found on the website

https://colab.research.google.com/notebooks/intro.ipynb

OPeNDAP and netCDF

- OPeNDAP (Open-source Project for a Network Data Access Protocol) enables the use of data from a remote server without the need of downloading the data files.
- Data can be accessed using a URL, and is stored in binary form.
- Offers sophisticated subsampling capabilities but with the interfaces available in python, Java, Matlab etc. , ability to extract the data in a more robust manner.
- Unidata's Network Common Data Form (netCDF) supports the creation, access, and sharing of array-oriented scientific data.
- Datasets consists of multi-dimensional blocks of numbers associated with a variable and each axis is ordered with the numbers with units.

More information and documentation can be found on the website

OPeNDAP™
Advanced Software for Remote Data Retrieval

https://www.opendap.org/

unidata
Data Services and Tools for Geoscience

https://www.unidata.ucar.edu/software/netcdf/index.html

Dataset

Reading the OPeNDAP Link. The URL is read as a netCDF file and it functions in the same way.

```python
from netCDF4 import Dataset

data = Dataset('http://apdrc.soest.hawaii.edu:80/dods/public_data/Reanalysis_Data/ERA5/monthly_3d/Specific_humidity')
```

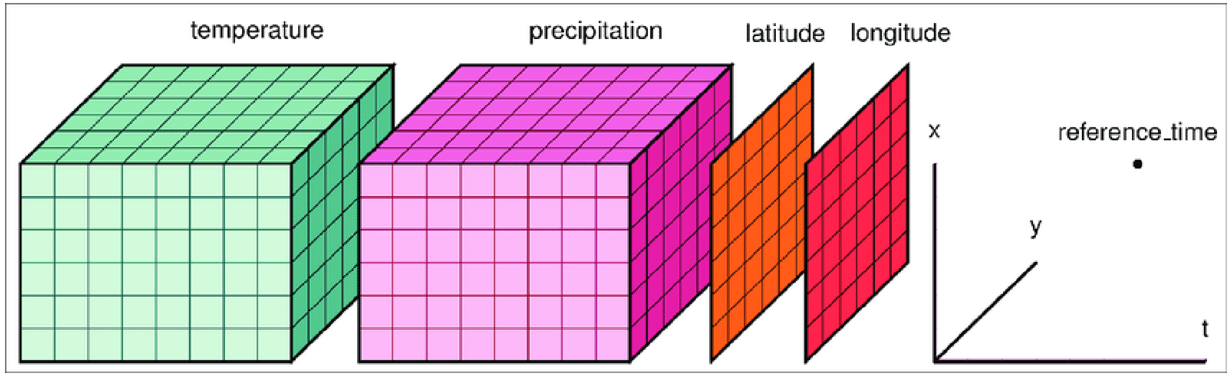
```
data

<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_CLASSIC data model, file format DAP2):
    title: ERA5 monthly averaged reanalysis Specific humidity on pressure levels
    Conventions: COARDS
GrADS
    dataType: Grid
    documentation: http://apdrc.soest.hawaii.edu/datadoc/ecmwf_era5.php
    history: Tue Jul 21 13:02:23 HST 2020 : imported by GrADS Data Server 2.0
    dimensions(sizes): lat(721), lev(32), lon(1440), time(498)
    variables(dimensions): float64 time(time), float64 lev(lev), float64 lat(lat), float64 lon(lon), float32 shum(time, lev, lat, lon)
    groups:
```

Link to access the Documentation of the dataset.

Dataset last updated.

A 4 dimensional array with the dimension variables: “lat” (latitude) , “lon” (longitude) , “lev” (level) and “time” (reference time).

The respective resolution of the variables are mentioned in the dataset along with the size of array.



Hoyer, Stephan & Hamman, Joseph. (2017). xarray: N-D labeled Arrays and Datasets in Python. Journal of Open Research Software. 5. 10.5334/jors.148.

Extracting Time Series Data

Importing all necessary Modules.

Storing all available latitude and longitude data of the dataset into the variables “lat” and “lon”.

Reading the OPeNDAP Link. The URL is read as a netCDF file and it functions in the same way.

```python
from netCDF4 import Dataset
import numpy as np
import pandas as pd

# Reading in the netCDF file
data = Dataset('http://apdrc.soest.hawaii.edu:80/dods/public_data/Reanalysis_Data/ERA5/monthly_3d/Specific_humidity')

# Storing the lat and lon data into the variables
lat = data.variables['lat'][:]
lon = data.variables['lon'][:]

# Storing the lat and lon of Western Ghats into variables
lat_WestnGhats =  10.1667
lon_WestnGhats =  77.0667

# Squared difference of lat and lon
sq_diff_lat = (lat - lat_WestnGhats)**2
sq_diff_lon = (lon - lon_WestnGhats)**2

# Identifying the index of the minimum value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
```

Entering the Latitude and longitude value of the point of consideration. The units for this particular dataset is in “N (north)” for latitude and “E (east)” for longitude.

```python
data.variables['lat']
```

```
<class 'netCDF4._netCDF4.Variable'>
float64 lat(lat)
    grads_dim: y
    grads_mapping: linear
    grads_size: 721
    units: degrees_north
    long_name: latitude
    minimum: -90.0
    maximum: 90.0
    resolution: 0.25
unlimited dimensions:
current shape = (721,)
```

```python
data.variables['lon']
```

```
<class 'netCDF4._netCDF4.Variable'>
float64 lon(lon)
    grads_dim: x
    grads_mapping: linear
    grads_size: 1440
    units: degrees_east
    long_name: longitude
    minimum: 0.0
    maximum: 359.75
    resolution: 0.25
unlimited dimensions:
current shape = (1440,)
```

Generating the index values corresponding to the minimum of the squared difference calculated.

Calculating the squared difference values from the point of consideration, to find the nearest point available from it.

Extracting Time Series Data (continued)

```
data.variables['shum']

<class 'netCDF4._netCDF4.Variable'>
float32 shum(time, lev, lat, lon)
    _FillValue: 9.999e+20
    missing_value: 9.999e+20
    long_name: specific humidity [kg kg**-1]
unlimited dimensions:
current shape = (498, 32, 721, 1440)
filling off
```

Accessing "shum" variable from the Dataset.

```
data.variables['time']

<class 'netCDF4._netCDF4.Variable'>
float64 time(time)
    grads_dim: t
    grads_mapping: linear
    grads_size: 498
    grads_min: 00z01jan1979
    grads_step: 1mo
    units: days since 1-1-1 00:00:0.0
    long_name: time
    minimum: 00z01jan1979
    maximum: 00z01jun2020
    resolution: 30.436619
unlimited dimensions:
current shape = (498,)
filling off
```

```python
# Storing the specific humidity data into the variable
Humidity = data.variables['shum']

# Creating and storing the data in an empty pandas dataframe
date_range = pd.date_range(start='1979/01/01', periods=498, freq='M')
df = pd.DataFrame(0, columns = ['shum'], index = date_range)
dt = np.arange(0,498)

for time_index in dt:
    df.iloc[time_index] = Humidity[time_index,0,min_index_lat ,min_index_lon]

# Saving the time series into a csv
df.to_csv('Specific_Humidity_WestnGhats.csv')
```

Generating a DataFrame with a date range of 498 periods as the index and a column with the name "shum".

The Specific Humidity values are accessed and updated in the dataframe accordingly.

	Unnamed: 0	shum
0	1979-01-31	0.012415
1	1979-02-28	0.013899
2	1979-03-31	0.015887
3	1979-04-30	0.018742
4	1979-05-31	0.020681
...	...	...
493	2020-02-29	0.013654
494	2020-03-31	0.016086
495	2020-04-30	0.018078
496	2020-05-31	0.019839
497	2020-06-30	0.020434

DataFrame generated has been saved in csv format .The file will be downloaded in the path where you are running your notebook.

Plotting Time Series Data

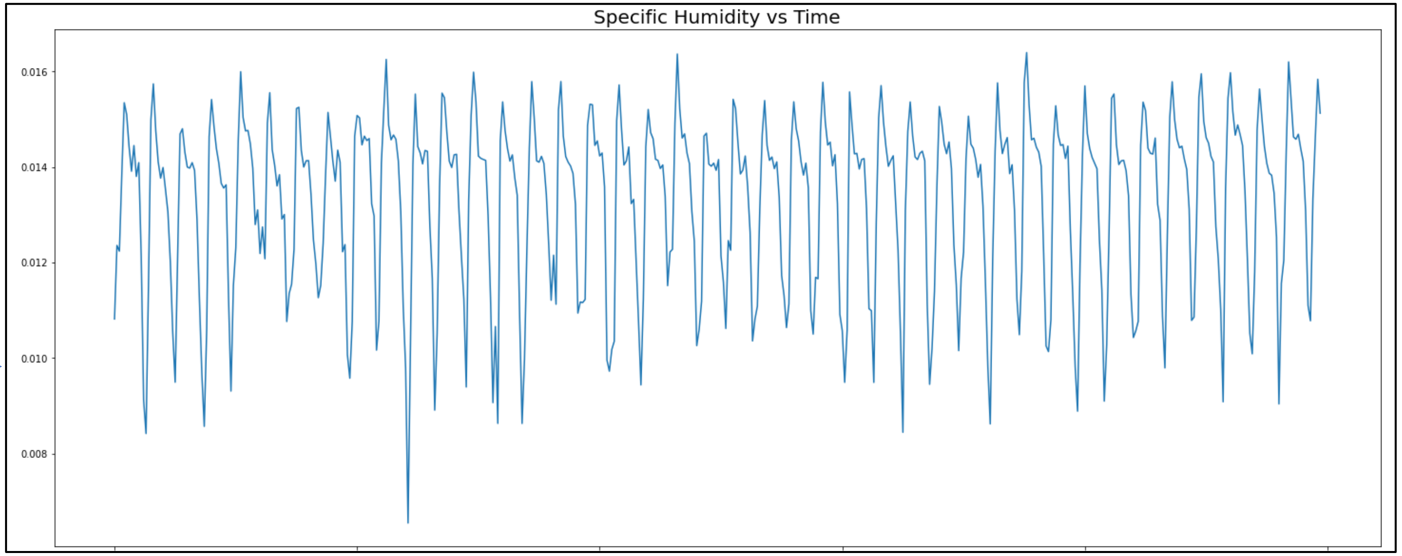
Importing all necessary Modules.

```
import pandas as pd
from matplotlib import pyplot as plt

data = pd.read_csv('Specific_Humidity_WestnGhats.csv')

plt.figure(figsize=(25,10))
plt.plot(data.index, data.shum)
plt.title('Specific Humidity vs Time', fontsize='20')
plt.show()
```

Reading data from a csv file.

Note: csv file should be present in the same path as the notebook

Plotting data with x-axis as the index values and the y-axis as humidity values.



Plotting Data on Map (Basemap)

Importing all necessary Modules.

```python
from mpl_toolkits.basemap import Basemap, cm
from netCDF4 import Dataset, date2index
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np

date = datetime(2003,1,1,0)

# open dataset.
dataset = Dataset('http://apdrc.soest.hawaii.edu:80/dods/public_data/satellite_product/MODIS_Aqua/4km_day')
timevar = dataset.variables['time']
timeindex = date2index(date,timevar)


uc = dataset.variables['sst'][timeindex,:].squeeze()
lats = dataset.variables['lat'][:]
lons = dataset.variables['lon'][:]
lons, lats = np.meshgrid(lons,lats)

# create Basemap instance.
fig = plt.figure(figsize=(12,9))
m = Basemap(projection='mill',llcrnrlat=5.734,urcrnrlat=24.3777,llcrnrlon=78.8982,urcrnrlon=95.0488, resolution="c")
m.drawcoastlines()
im1 = m.pcolormesh(lons,lats,uc,shading='flat',cmap=plt.cm.jet,latlon=True)
cb = m.colorbar(im1,"bottom", size="5%", pad="1%")
ax.set_title('SST for 2003-01-01')
plt.show()
```

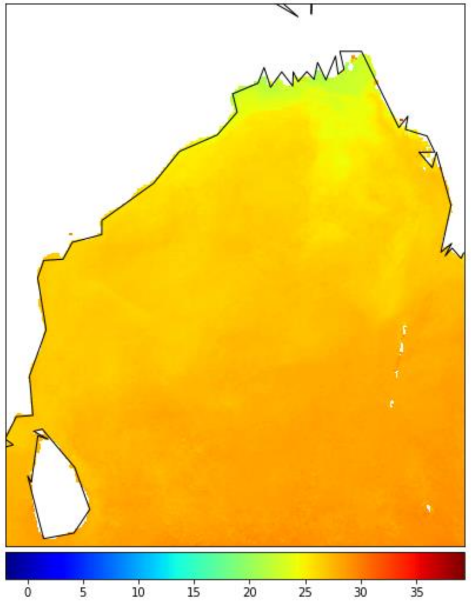
Reading the OPeNDAP Link.

Generating the index of the time at which we need to plot the data.

Storing data in respective variables

Plotting the data on map

OUTPUT



More information and documentation can be found on the website of basemap

https://matplotlib.org/basemap/