* Python (Data science)
* Essential basic function for data science
* NUmpy
* pandas
* matplotlib
* ?: mark to find this docstring
* ??: Accessing Source Code with ??
* The benefit of ***%timeit*** is that for short commands it will automatically perform multiple

runs in order to attain more robust results.

* ***%%timeit*** for multiple line.
* In and Out.
* ls list of all files
* pwd present working directory
* echo ‘ ’ prints the given content between the quotation.
* %xmode magic function, allows you to control the amount of information printed when the exception is raised
* %xmode takes one argument (Plain,verbose,Context,minimal)
* Context
* Plain is more compact and gives less information
* Verbose adds some extra information, including the arguments to any

functions that are called

* Minimal gives direct exception name with out informing why exception has been occurred.
* Argument may vary if we are not specifying the name of the argument.

ipdb prompt lets you

* Ipdb prompt lets you explore the current state of the stack, explore the available variables.

Numpy:

* In c we will declare a variable as follows

Int a=10;

//where a is pointing to the memory location containing the value of 10.

* But in python it is completely different

a=10

As python is dynamically typed it will take the value stored in the a as integer but in python a is not only pointing to the value 10,but it also contains

• ob\_refcnt, a reference count that helps Python silently handle memory allocation

and deallocation

• ob\_type, which encodes the type of the variable

• ob\_size, which specifies the size of the following data members

• ob\_digit, which contains the actual integer value that we expect the Python variable

to represent

PyObject\_HEAD is the part of the structure containing the reference count, type

code, and other pieces mentioned before.

Numpy functions(Routines):

* Import numpy as np
* np.zeros(shape,dtype) : creates a numpy array with zeros m\*n
* np.ones(shape,dtype) :creates a numpy array with ones
* np.full(value,shape,dtype) : creates a numpy array with specified value of specified shape
* np.arange(start,stop,step):creates a numpy array with start until stop with specified step value.
* np.linspace(start,stop,count):creates a numpy array with start and stop of length =count.
* np.random.random(size):creates a numpy array with random values between 0 to 1 of specified shape.
* np.random.randint(low,high,(size)):creates a numpy array with random values between low and high of specified shape.
* np.eye(N) : creates a numpy array of shape NxN containing diagonal elements as 1(default type is float).

Numpy datatypes:

Data type Description

* bool\_ Boolean (True or False) stored as a byte
* int\_ Default integer type (same as C long; normally either int64 or int32)
* intc Identical to C int (normally int32 or int64)
* intp Integer used for indexing (same as C ssize\_t; normally either int32 or int64)
* int8 Byte (–128 to 127)
* int16 Integer (–32768 to 32767)
* int32 Integer (–2147483648 to 2147483647)
* int64 Integer (–9223372036854775808 to 9223372036854775807)
* uint8 Unsigned integer (0 to 255)
* uint16 Unsigned integer (0 to 65535)
* uint32 Unsigned integer (0 to 4294967295)
* uint64 Unsigned integer (0 to 18446744073709551615)
* float\_ Shorthand for float64
* float16 Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
* float32 Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
* float64 Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
* complex\_ Shorthand for complex128
* complex64 Complex number, represented by two 32-bit floats
* complex128 Complex number, represented by two 64-bit floats

Numpy array attributes:

* np.ndim(a) or a.ndim : which gives dimensions of an specified array here array is a.
* np.shape(a) or a.shape :which gives shape of an numpy array.
* np.size(a) or a.size : which gives size of an array i.e number of elements in an array.
* np.itemsize(a) or a.itemsize: which gives the size of each item in an array.
* np.bytes(a) or a.nbytes: (itemsize \* size)

Numpy array indexing and slicing:

* Acessing the numpy array elements is same as list elements(specifying the 0-based index and also negative based index)

a=np.array([1,2,3],dtype=np.int)

a[0]=1=a[-3]

a[1]=2=a[-2]

a[2]=3=a[-1]

* a=np.arange(0,9,1).reshape(3,3)

print(a) [[0 1 2][3 4 5][6 7 8]]

print(a[0]) [0 1 2]

print(a[1]) [3 4 5]

print(a[2]) [6 7 8]

print(a[:,0]) [0 3 6]

print(a[:,-1]) [2 5 8]

print(a[::-1]) [[6 7 8][3 4 5][0 1 2]]

print(a[::-1,::-1]) [[8 7 6] [5 4 3][2 1 0]].

Subarrays as no copies:

array slices is that they return *views* rather than *copies* of the array data. This is one area in which

NumPy array slicing differs from Python list slicing: in lists, slices will be copies

|  |  |
| --- | --- |
| List | Numpy array |
| a=[[1,2,3],[6,7,9],[0,0,0]]#list  print(a)  b=a[2]  b[2]=3  print(b)  print(a)  output:  [[1, 2, 3], [6, 7, 9], [0, 0, 0]]  [0, 0, 3]  [[1, 2, 3], [6, 7, 9], [0, 0, 3]] | a=np.array([[1,2,3],[6,7,9],[0,0,0]])#array  print(a)  b=a[2]  b[2]=3  print(b)  print(a)  output:  [[1 2 3]  [6 7 9]  [0 0 0]]  [0 0 3]  [[1 2 3]  [6 7 9]  [0 0 3]] |

Array Concatenation and splitting:

Array concatenation:

* Concatenation, or joining of two arrays in NumPy, is primarily accomplished through the routines
  + np.concatenate (working with similar dimensions)
  + np.vstack(vertical) (working with different dimensions)
  + np.hstack(horizontal) (working with different dimension)
* np.concatenate takes a tuple or list of arrays as its first argument.

Splitting of arrays:

* The opposite of concatenation is splitting.
  + np.split(arr,section)
  + np.hsplit (arr,section)
  + np.vsplit (arr,section)

For each of these, we can pass a list of indices

Exploring NumPy’s UFuncs:

Ufuncs exist in two flavors:

* unary ufuncs, which operate on a single input,
* binary ufuncs, which operate on two inputs.
  + + np.add ()
  + - np.subtract()
  + - np.negative ()
  + \* np.multiply()
  + / np.divide()
  + // np.floor\_divide()
  + \*\* np.power()
  + % np.mod()
* For complex data absolute value returns the magnitude
* Absolute function returns array with only positive numbers.

Functions:

sin(theta) np.sin(theta))

cos(theta) np.cos(theta))

tan(theta) np.tan(theta))

arcsin(x) np.arcsin(x))

arccos(x) np.arccos(x))

arctan(x) np.arctan(x))

e^x np.exp(x))

2^x np.exp2(x))

3^x np.power(3, x))

ln(x) np.log(x))

log2(x) np.log2(x))

log10(x) np.log10(x))

exp(x) – 1 np.expm1(x))

log(1 + x) np.log1p(x))

Aggregate functions:

np.sum Compute sum of elements

np.prod Compute product of elements

np.mean Compute mean of elements

np.std Compute standard deviation

np.var Compute variance

np.min Find minimum value

np.max Find maximum value

np.argmin Find index of minimum value

np.argmax Find index of maximum value

np.median Compute median of elements

np.percentile Compute rank-based statistics of elements

np.any Evaluate whether any elements are true

np.all Evaluate whether all elements are true

axis=0 columns

axis=1 rows

Sorting:

Sorting is used to arrange the elements either in ascending order or descending order. As we know the famous sorting algorithms are selection, bubble, insertion sort and also bogosort is a sorting technique where sorting is based on the (randomized shuffle) which takes O(N x N!).

While np.sort() and np.argsort() are two sorting functions while sort() and argsort() uses quicksort

algorithm, though mergesortand *heapsort* are also available .while this function takes O(N x logN).

Partial Sorts: Partitioning

Sometimes we’re not interested in sorting the entire array, but simply want to find the *K* smallest values in the array. NumPy provides this in the np.partition function. np.partition takes an array and a number *K*; the result is a new array with the smallest *K* values to the left of the partition, and the remaining values to the right, in arbitrary order.

np.partition(arr,K)

Data Manipulation with Pandas:

Pandas is a newer package built on top of NumPy, and provides an

efficient implementation of a DataFrame.

Pandas series Data:

Import pandas as pd

a=pd.Series([1.2,3,9.0])

print(a)

o/p:

1. 1.2
2. 3
3. 9.0

float64

* a.values retrive the data of the series.
* a.index gives the information about index values
* NumPy array has an implicitly defined integer index used to access the values, the Pandas Series has an *explicitly defined* index associated with the values.
* Data can be a scalar in pandas Series.
* Data can be a dictionary, in which index defaults to the sorted dictionary keys.

pd.Series(data=None,index=None,dtype=None)

* data : by default data is none we have to specify the data(list,tupe,dictonary).
* index : by default it is zero based index but if we mention any index values it will map the data with the specified values.
* dtype : by defult dtype is none the data may vary depends upon data.

Pandas Dataframe data:

* If a series is analog of one-dimensional array with flexible indices,a Dataframe is analog of a two-dimensional array with both flexible rows indices and flexible column indexes.
* We can even use noncontiguous or nonsequential indices.
* If some keys in the dictionary are missing, Pandas will fill them in with NaN(Not a Number).

Pandas Index Object:

* This Index object is an interesting structure in itself, and it can be thought of either as an immutable arrayor as an ordered set(technically a multiset, as Index objects may contain repeated values).Those views have some interesting consequences in the operations available on Index.
* Index object has many attributes familiar with Numpy arrays.
* Index Object indices are immutable i.e we can not modify the data .
* The Index object follows many of the conventions used by Python’s built-in set data structure, so that unions, intersections , differences, and other combinations.

Data Indexing and Selection :

* Notice that when you are slicing with an
  + explicit index (i.e., data['a':'c']), the final index is *included* in the slice.
  + implicit index (i.e., data[0:2]), the final index is *excluded* from the slice.
* Indexer
  + loc : loc attribute allows indexing and slicing that always references the explicit index.
  + iloc : iloc attribute allows indexing and slicing that always refernces the implic index.
  + ix : ix is the hybrid of two, in Series it is equivalent to standard based indexing.

Operating in data in pandas :

* numpy ufuns will work on pandas series and dataframe data.

|  |  |  |
| --- | --- | --- |
| Python operators | Pandas methods | |
| + | add() | |
| - | sub(),subtract() | |
| \* | mul(),multiply() | |
| / | div(),truediv(),divide() | |
| // | floordiv() | |
| % | mod() |
| \*\* | Pow() | | |

Handling with missing data:

* the real world data is rearly clean and homogenous.
* different data sources may indicate missing data in different ways.
* Generally missing data will refer NA,Nan,null.
* The number of strategies have been developed to identify the missing data.
  + Mask Vlaue :globally indicates the missing value.
  + Sentinel Value :indicates the missing entry.
* The result of arithmetic operations with Nan will be Nan no matter what is the operator.
* Pandas handling of NA’s by type:

|  |  |  |
| --- | --- | --- |
| Type class | Converting | NA sentinel values |
| float | No conversion | np.Nan |
| object | No conversion | None,np.Nan |
| integer | Convert into floating | np.Nan |
| boolean | Convert into object | None,np.Nan |

string data will always stored in object dtype.

* Operating on NULL values
  + There are several useful methods to detect,remove and replacing the null values in pandas datastructure.
    - Isnull() generate Boolean masked data
    - notnull() opposite if isnull
    - dropna() return a filtered data.
      * df.dropna(axis=’rows|columns‘,thresh=value)
        + thresh: threshold which specify the minimum data.
    - fillna() return copy of data after filling with missing data.
      * df.fillna(values).
      * df.fillna(method=’ffill’). forward fill
      * df.fillna(method=’bfill’). backward fill

Hierarchal indexing:

* it is also know as multi indexing.

matplotlib

Visualization with matplotlib:

* One of Matplotlib’s most important features is its ability to play well with many operating systems and graphics backends.
* Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish.
* This cross-platform, everything-to-everyone approach has been one of the great strengths of Matplotlib.
* It has led to a large userbase, which in turn has led to an active developer base and Matplotlib’s powerful tools and ubiquity within the scientific Python world.

1. plt.show() starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure or figures.
2. plt.show() command is only used once for each python session.
3. plt.figure() command is used to plot the figure and using this we can create an separate image of the plot.

fig=plt.figure()

fig.savefig(‘nameofthefigure with extension’)

Two interface for the price of one:

* plt.figure() creates a plot figure
* plt.subplot(rows,columns,panel\_number) : which creates a multiple graphs with in the same session.
* plt.gcf(): get current figure.
* plt.gca(): get current axes.

Using object oriented approach:

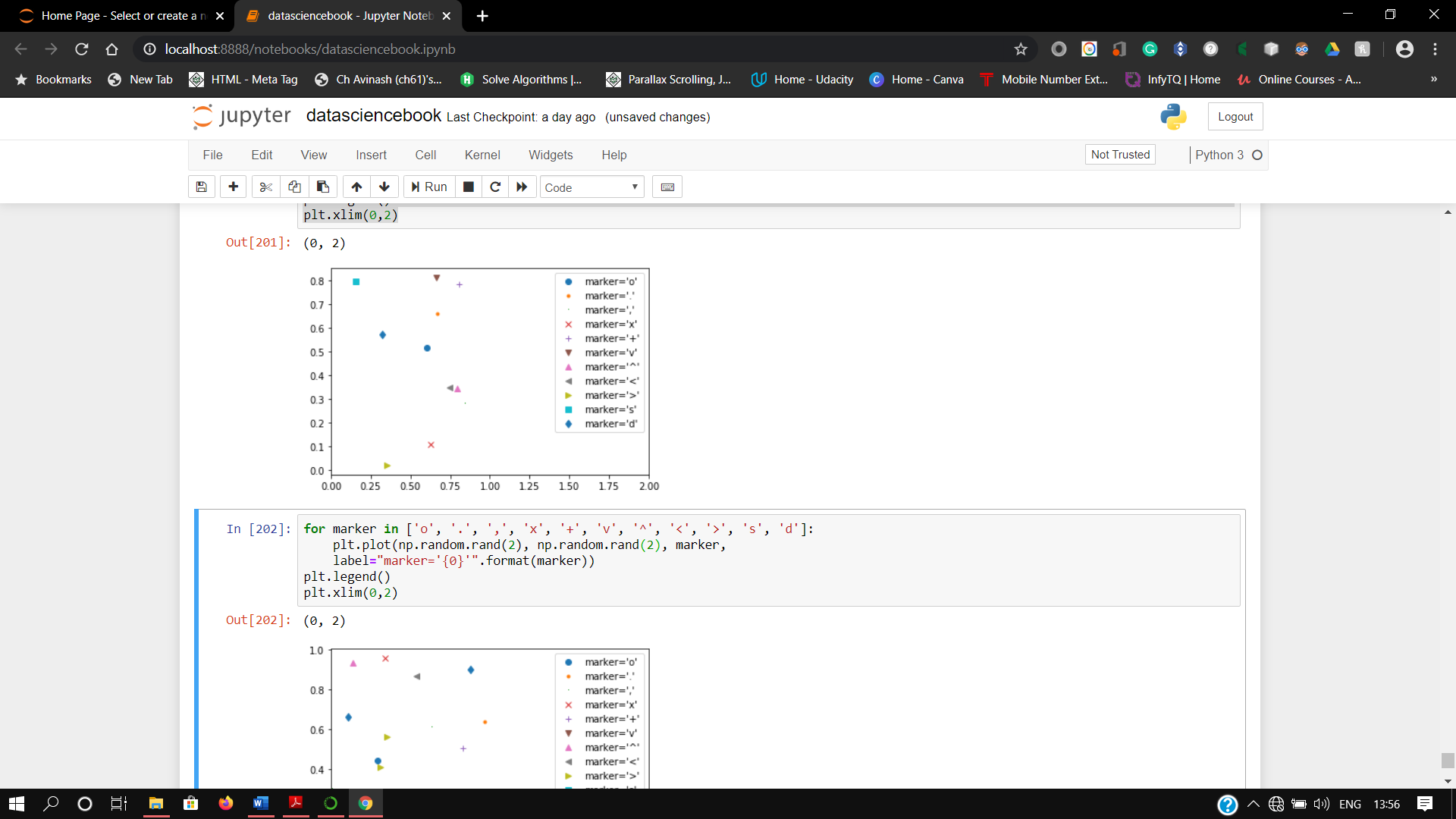
Here we will create a subplots by using following syntax

fig,axis=plt.subplots(nrows,ncolumns)

axis[row][column].plot()

* plot() function arguments.
  + Color : color is a keyword which accepts a string of virtually imaginable colors.
    - specify color by name
    - short color code (rgbcmyk) CMYK (Cyan/Magenta/Yellow/blacK)
    - Grayscale between 0 and 1
    - Hex code (RRGGBB from 00 to FF)
    - RGB tuple, values 0 and 1
    - all HTML color names supported
  + style: line styles
    - solid ‘-‘
    - dashed ‘- -'
    - dashdot ‘-.’
    - dotted ‘:’
  + ‘-g’ solid green
  + ‘- -r’ dashed red
  + ‘-.y’ dashdot yellow
  + ‘:b’ dotted blue
* Axes limits:
  + plt.xlim(xmin,xmax)
  + pt.ylim(ymin,ymax)
  + The plt.axis() method allows you to set the x and y limits with a single call, by passing a list that specifies [xmin, xmax, ymin,ymax].
* plt.title():we can specify the title of the graph.
* plt.xlabel():to specify label of x axis.
* plt.ylabel():to specify label of y axis.
* When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type
  + plt.legend()
* rather than calling all functions individually,it is often more convenient to use the ax.set() method to set all these properties at once.

Simple Scatter plots:

* Another commonly used plot type is the simple scatter plot, a close cousin of the line plot.
* Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.
* We can create a scatter graph by using plt.plot() function itself.
* Along with plt.plot() we can also create scatter plot with plt.scatter() function.

Density and Contour plots:

* Sometimes it is useful to display three-dimensional data in two dimensions usingcontours or color-coded regions.

* There are three Matplotlib functions that can behelpful for this task:
  + plt.contour for contour plots,
  + plt.contourf for filled contour plots
  + plt.imshow for showing images.
* A contour plot can be created with the plt.contour function. It takes three arguments :a grid of *x* values, a grid of *y* values, and a grid of *z* values. The *x* and *y* values represent positions on the plot, and the *z* values will be represented by the contour levels.
* Perhaps the most straightforward way to prepare such data is to use the np.meshgrid function, which builds two-dimensional grids from one-dimensional arrays.

Histogram binning and density:

* A simple histogram will create a great understanding of the dataset.
* hist function has many objects to tune the calculation and the display.

Two dimensional histogram:

* Histograms in two dimensions by dividing points among two dimensional bins.

plt.legend:

* loc=’best,upper right,upper left,lower left,lower right,right,center left,center right,lower center,upper center,center.
* frameon=True/False.
* (fancybox) or add a shadow, change the transparency.
* (alpha value) of the frame, or change the padding around the text.

Customizing colorbars:

* Choosing color bars
  + Sequential colormaps :These consist of one continuous sequence of colors (e.g., binary or viridis).
  + Divergent colormaps : These usually contain two distinct colors, which show positive and negative deviations from a mean (e.g., RdBu or PuOr).
  + Qualitative colormaps :These mix colors with no particular sequence (e.g., rainbow or jet).

3diemensional figures:

* We enable three-dimensional plots by importing the mplot3d toolkit.

from mpl.toolkits import mplot3d

* Once mplot3d is imported we can create a 3 dimensional figures.
* for plotting 3 d image we require 3 variables x,y,z.
* where x and y are the input and z is the resultant for our assumption.

3D contour plots:

* for creating a contour plots first we have to a meshgrid with 2 variables .
* using these 2 variables we will compute the 3rd variable as result and with the help of

contour3d() function we can plot the figure.

* ax=plt.axes(projection=’3d’)
* ax.set\_xlabel():to set x label
* ax.set\_ylabel():to set y label
* ax.set\_zlabel():to set z label
* ax.set\_title():used to set the title of the figure.
* sometimes it is very difficult to analyse the figure in default coordinates to over come this matplotlib contain function view\_init()
  + there are two arguments view\_init(elevation,azimuth)
    - elevation=60 (that is,60 degrees above the *x*-*y* plane)
    - azimuth=azimuth of 35 degrees (that is, rotated 35 degrees counter-clockwise about the *z*-axis).
* Two other types of three-dimensional plots that work on gridded data are wireframes

and surface plots.

ax.plot\_wireframe()

ax.plot\_surface()