



PG TECH WEEKEND

WnCC

Debriefing Data



IIT BOMBAY





Session Objectives



GET STARTED WITH EXPLORATORY DATA ANALYSIS

See how we can extract valuable information from data and use it to infer stuff



LEARN LIBRARIES FOR DATA HANDLING

Numpy and Pandas to filter and manage huge repositories of data



VISUALISE EXTRACTED DATA

Using libraries like `matplotlib` to plot and find visual correlations between data samples

Google Colab : The Basics

Hosted jupyter notebook service

ML/DS without having to worry
about computational power

Importing libraries - Simplified



Loading data in Colab

Upload your data to google drive

The most important step: Mount your drive

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

```
Mounted at /content/drive
```

NUMPY

WHY SHOULD I USE NUMPY? AM I NOT DOING
WELL ENOUGH WITH LOOPS AND LISTS?

NO.

Vectorisation over loops.
Numpy arrays over lists.

Data analysis is computation-heavy. Using
libraries like numpy optimises a lot of
operations - for example matrix multiplication.

Numpy arrays also take less memory than
python lists.



Installation

We'll be using these Libraries

```
pip install numpy
```

```
pip install scipy
```

```
pip install pandas
```

```
pip install matplotlib
```



Getting started with numpy

with help from our trusty friend, python lists!

* `numpy.array(object)`

Where `object` is array-like (a python list, tuple, a nested python list or tuple) returns a numpy array constructed from this object


We can represent matrices using numpy arrays, where the object is a list of lists, like `[[1, 2], [3, 4]]`. The array formed is a 2D array, and we can make arrays of `n` dimensions similarly. These arrays are of the type `<numpy.ndarray>`



arange() and linspace()



Similar to the `range(start, stop, step)` in python

`np.arange()` returns a numpy array of evenly spaced values in the given interval `[start, stop)`. Precision over the step



When the step sizes are non-integer, such as 0.1, it is best to use **`linspace`** instead of `arange`.

`np.linspace(start, stop, num)` returns a numpy array of evenly spaced numbers over a specified interval, `[start, stop]`. Precision over the end values






ones and zeros - shapes



`np.zeros(shape)` returns an array of specified shape, filled with zeros.

`np.ones(shape)` returns an array of specified shape, filled with ones.

`np.reshape(array, newshape)` or `array.reshape(newshape)` changes the shape of an array without changing its data.



This gives us good matrix, or an N-dimensional numpy array of a shape of choice to play and experiment with.



A decorative border surrounds the slide, featuring a variety of hand-drawn, colorful shapes and patterns. At the top, there's a red square, a blue swirl, a yellow diamond, a red spiral, a green triangle, a red concentric circle, a pink semi-circle, and a blue L-shape. At the bottom, it includes a red swirl, a yellow spiral, a blue L-shape, a red swirl, a green flower-like shape, a yellow oval, and several red and yellow stars.

Random Numbers

`np.random.rand(shape)`: returns an array of specified shape, filled with random numbers in $[0, 1)$.

`np.random.randint(low, high=None, size=None, dtype=int)`: Only **low** is a compulsory parameter. Returns an array of size = **size** filled with random integer in range $[low, high)$. If **high** is not given then returns in the range $[0, low)$.



Indexing and Selection

We use the **square bracket** to select a part of the ndarray object.

`arr[8]` # gives the element at index 8

`arr[:5]` # returns the array from the start to one before the end index

`arr[3:]` # returns the array from the start index (included) to the last one

INDEXING FOR 2D ARRAYS

Square brackets again. This time, the elements might be arrays themselves

`matrix[i]` # returns the *i*th row of the 2D matrix

`matrix[i][j]` # returns the element at row *i*, column *j*

`matrix[i:]` # returns all the rows after and including the *i*th row

`matrix[1:5, 3:6]` # returns a 4x3 submatrix, with elements from the 1st to the 4th row and 3rd to the 5th column

`matrix[:i, :j]` # similar, the *i*x*j* submatrix in the top left, with elements whose rows < *i* and columns < *j*



CONDITIONAL SELECTION

`myarray <operator> value*`
returns a boolean array of
shape similar to `myarray` whose
elements are the condition
applied to every element in
`myarray`

```
>>> bool_arr =  
np.array([1, 2, 3], [4, 5, 6]) > 3  
>>> bool_arr  
array([False, False, False], [True, True,  
True])
```

*VALUE CAN BE ARRAY-LIKE, SEE BROADCASTING

`ndarray[boolean array]` returns
the elements of the array for
which the corresponding element
in the boolean array is true.

```
>>> arr = np.array([1, 2, 3, 4, 5],  
[6, 7, 8, 9, 10])  
>>> arr[arr%2==0]  
[2, 4, 6, 8, 10]
```

Arithmetic Operations

You can add two arrays if they are of the same shape
using just +

Matrix Multiplication:

If a and b are arrays, their matrix multiplication is done by
a@b

Elementwise Product:

Done using *



Broadcasting

With numpy you can apply functions for a single element for the entire array very efficiently.
Numpy does this automatically.

```
def addOne(a):  
    return a+1  
  
x = numpy.ones(15, 40)  
y = addOne(x)
```

Each element in y would be 1 + the corresponding element in x





Numpy Mathematical Functions

Numpy provides some inbuilt mathematical functions

Some useful examples include:



All trigonometric functions like `np.sin(angle)`, `np.cos(angle)`

`np.exp(x)` for exponentiation

`np.log(x)` for natural log

and many more...



Pandas





SERIES DATA TYPE

The series data type is quite similar to a numpy array. The difference is that a **Series** object can have axis labels, meaning it can be indexed by a label, instead of just a number.

```
pandas.Series(data, index)
```

data is array-like and contains the data stored in the series

index is array-like, of same length as data and contains the labels for the data values



Eg. **data** = [10, 20, 30]

index = ['A', 'B', 'C']



Thus the elements in data can be indexed by their corresponding labels too.

```
>>> my_series = pandas.Series(data=[1, 2, 3], index=['A', 'B', 'C'])
```

```
>>> my_series['A']
```

```
1
```





DATAFRAMES

DataFrames are a crucial data type in the Pandas library. They can be thought of as a bunch of Series objects put together to share the same **index**.

```
pandas.DataFrame(data, index, columns)
```

data is an ndarray or some other iterable and contains the data stored in the DataFrame

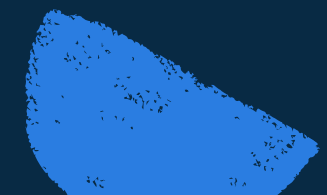

index is array-like, and specifies the indexing for the DataFrame

columns specifies the column labels for the data



```
Eg. data = [[90, 80], ['A', 'B']]  
index = ['Marks', 'Grade']  
columns = ['Alice', 'Bob']
```

Here columns are a labelled Series object which share the same indices, namely 'Marks' and 'Grade'





SELECTION OF ELEMENTS

To select a Series object, use **square brackets []** with the label of the column.

```
>>> my_dataframe[labels] # labels can be a list of column names
```

To remove a row or a column, we use the **drop(labels, axis) function**, where labels is the list of labels to remove, and axis specifies whether these labels are indices or of columns.



To select **rows**, we use the **.loc[labels]** syntax.

```
df.loc['Marks'] # the row with index 'Marks', returned as a Series object  
                with column labels as indices
```

```
df.loc['Marks', 'Alice'] # labels specifying both the row and the column  
df.loc[index-list, label-list] # part of the DataFrame, like indexing a 2D  
                                array
```





MISSING DATA

We can drop the rows or columns entirely which have a missing value.

`dropna(axis, how, thresh)` can be used.

Look at the Jupyter Notebook for parameter info.



CONDITIONAL SELECTION

Look at the Jupyter Notebook





INPUT AND OUTPUT



Pandas can read a variety of file types using its `pandas.read_` methods.

CSV FILES

```
my_dataframe = pandas.read_csv(Input_Filepath)
my_dataframe.to_csv(Output_Filepath)
```

EXCEL SHEETS

```
my_dataframe = pandas.read_excel(Input_Filepath, sheet_name)
my_dataframe.to_excel(Output_Filepath, sheet_name)
```



A decorative border surrounds the central text, featuring various colorful shapes and patterns. At the top, there is a red square, a blue wavy line, a yellow diamond, a red scribble, a green triangle, a red spiral, a pink semi-circle, and a blue arrow. At the bottom, there is a red wavy line, a yellow scribble, a blue L-shape, a red scribble, a green flower-like shape, a yellow oval, and several red and yellow stars.

Matplotlib

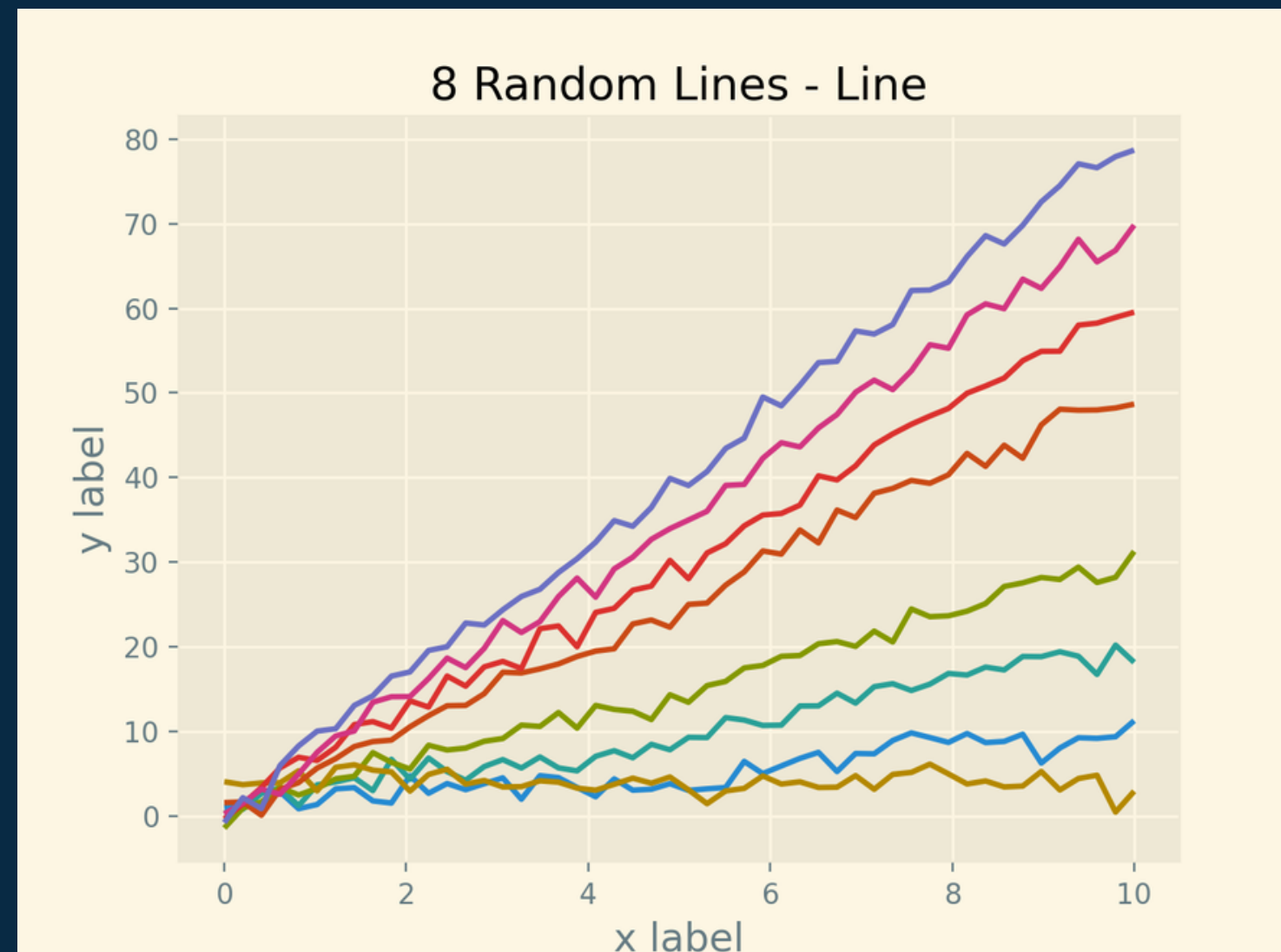
Make beautiful graphs to display your data

```
import matplotlib.pyplot as plt
```

PLOTTING DATA

Given two arrays for x and y coordinates,

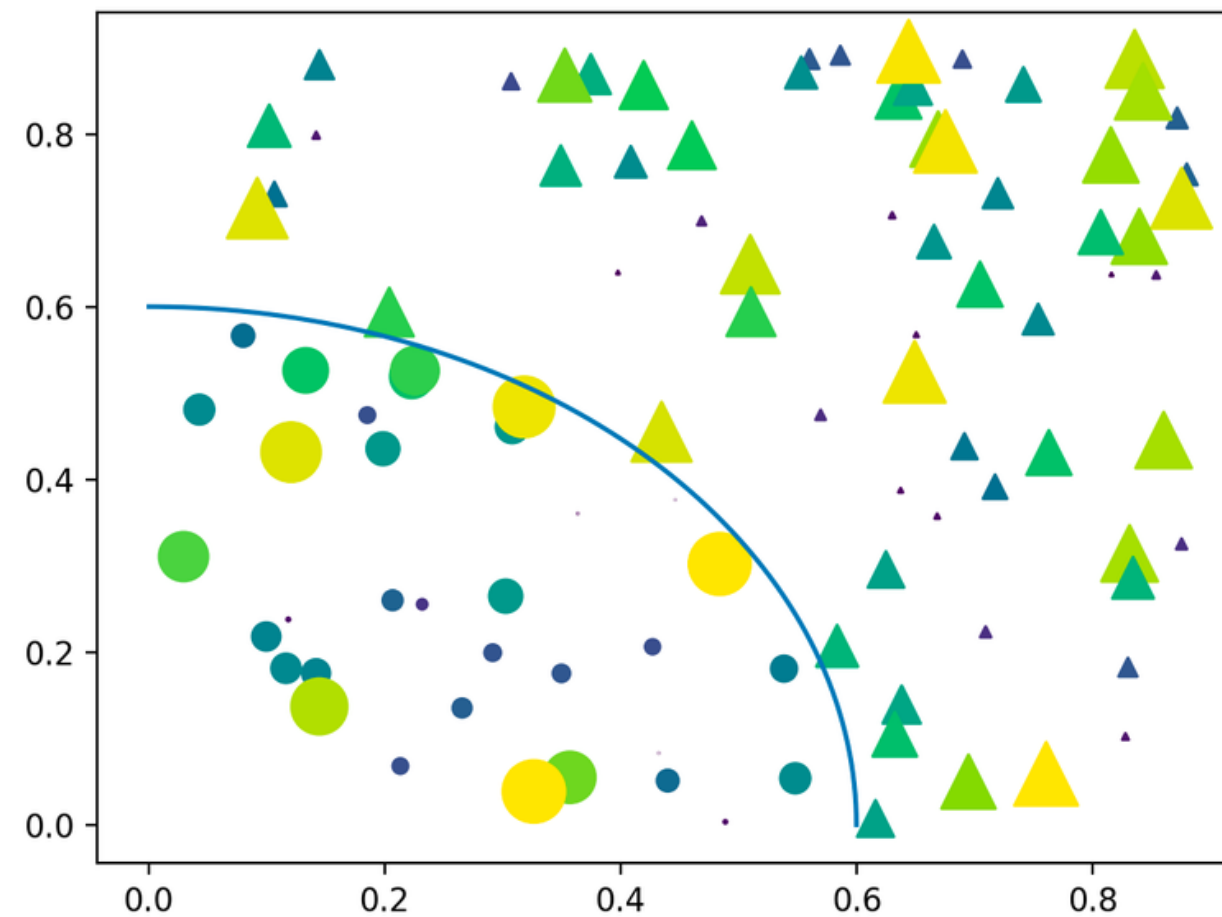
```
plt.plot(x, y)  
plt.show()
```



SCATTER PLOTS

plt.scatter(x, y)

Some better properties than plt.plot if you just want to plot some points



src = https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_masked.html

HISTOGRAMS

Use the `plt.hist()` function to plot a histogram

`plt.hist(x, bins, range, density)`

`x` is the data in form of an array

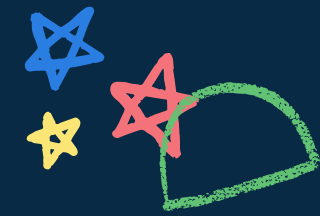
`bins` specifies the number of bins to divide the data into

`range` specifies the lower and upper range of the bins, can be used to narrow down data points

`density` when True, will draw and return the probability density : the histogram will be normalised such that the areas add to 1

Tell me and I forget, teach
me and I may remember,
involve me and I learn.

- Benjamin Franklin



Be Your Own Sherlock!

Invoke the detective in you

A scam has taken place at Dunder Mifflin and some funds have been embezzled. This wasn't the work of a single person.

You've been hired to bring the perpetrators to light.

Use your DS skills to deliver justice



Fin.

Until Next Time.

GO THROUGH THE EXAMPLES IN THE
NOTEBOOK, AND SUPPLEMENT THEM WITH
THE DOCS. YOU'LL GET THE HANG OF IT IN
NO TIME!