proschool
An ims Initiative

Python


Let me show you how to install packages

How to install packages in Anaconda 3

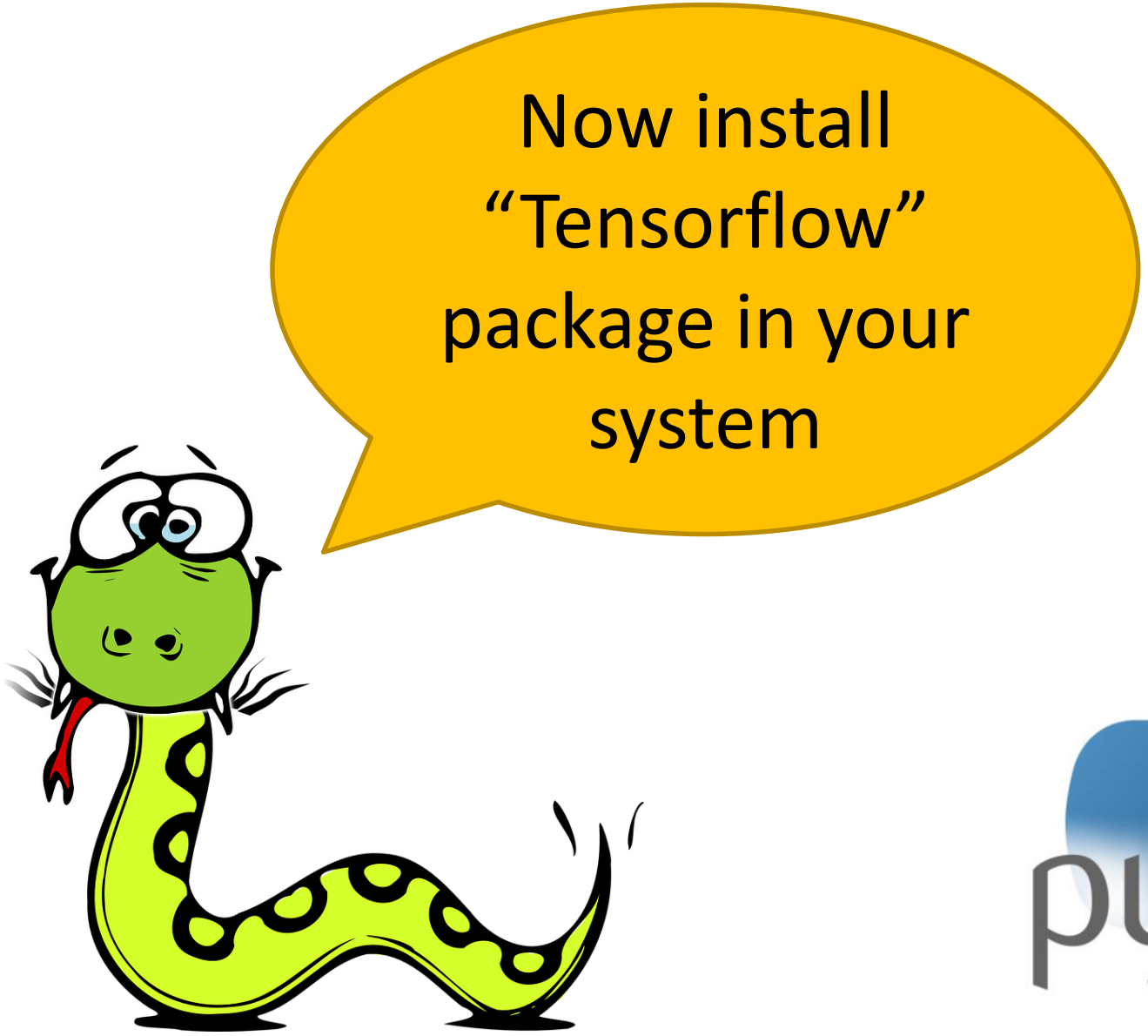
proschool
An ims Initiative

- Open command prompt in your system.
- Change the existing directory to the directory where Anaconda Scripts have been written
- Install packages with the help of -> “pip install <packagename>

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\hi>cd Anaconda3

C:\Users\hi\Anaconda3>cd Scripts

C:\Users\hi\Anaconda3\Scripts>pip install matplotlib
Requirement already satisfied: matplotlib in c:\users\hi\anaconda3\lib\site-pack
ages
Requirement already satisfied: numpy>=1.7.1 in c:\users\hi\anaconda3\lib\site-pa
ckages (from matplotlib)
Requirement already satisfied: six>=1.10 in c:\users\hi\anaconda3\lib\site-packa
ges (from matplotlib)
Requirement already satisfied: python-dateutil>=2.0 in c:\users\hi\anaconda3\lib
\site-packages (from matplotlib)
Requirement already satisfied: pytz in c:\users\hi\anaconda3\lib\site-packages (
from matplotlib)
Requirement already satisfied: cycler>=0.10 in c:\users\hi\anaconda3\lib\site-pa
ckages (from matplotlib)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\us
ers\hi\anaconda3\lib\site-packages (from matplotlib)

C:\Users\hi\Anaconda3\Scripts>
```

Directory has been changed

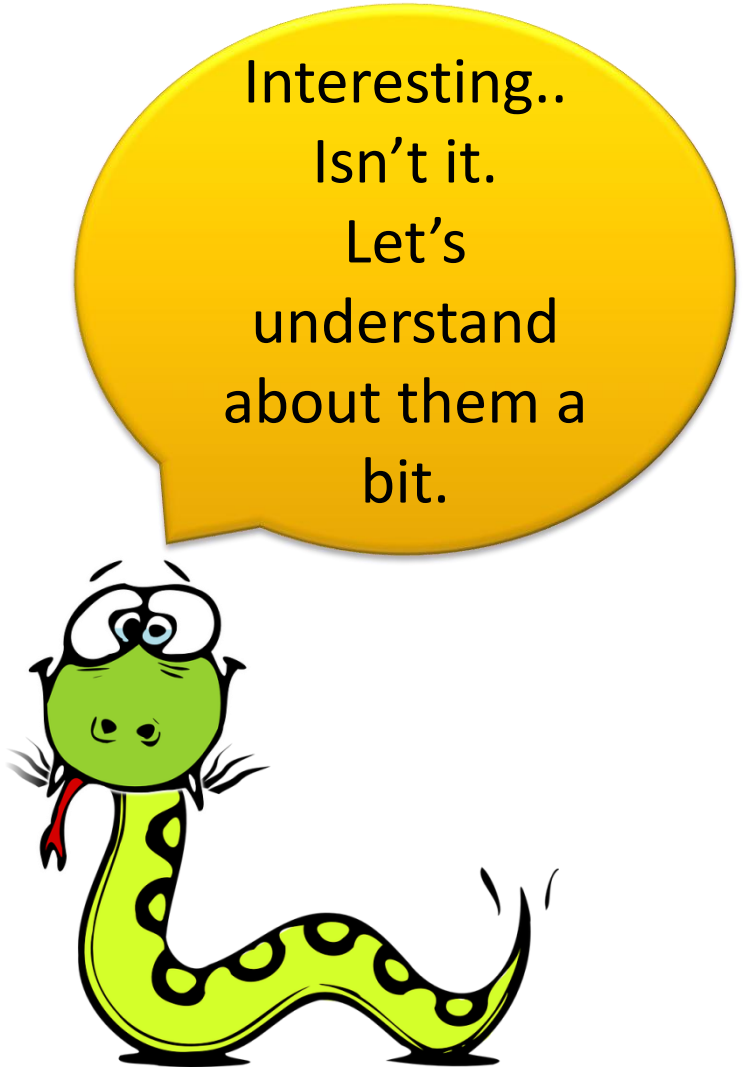
Using “pip”, I have tried to install package-matplotlib


Now install
“Tensorflow”
package in your
system


Now, we will
learn about some
useful packages
in Python.

Useful packages

- Numpy
- Pandas
- Matplotlib
- Scikit –Learn
- NLTK



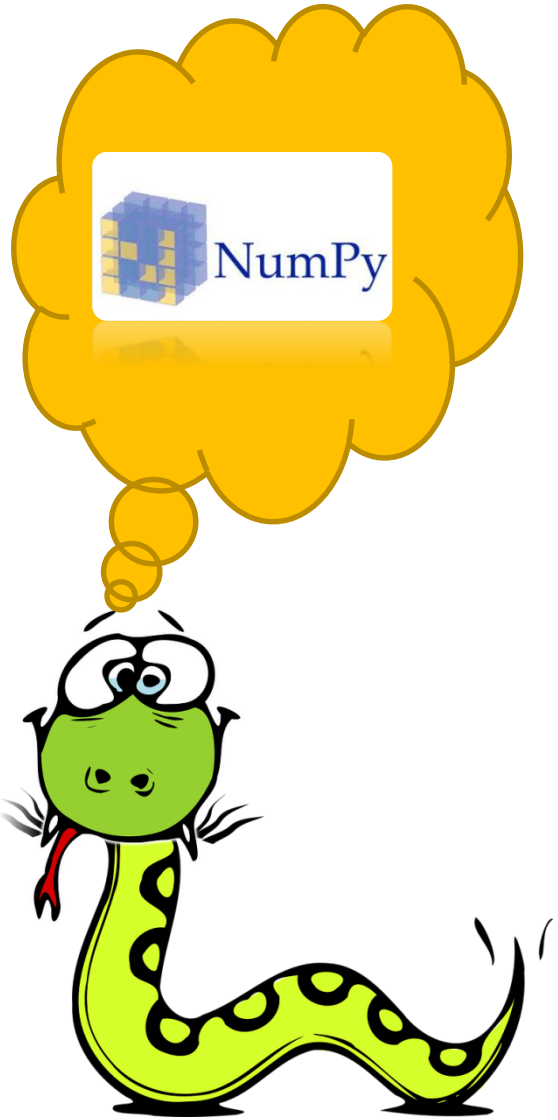
Numpy

NumPy (stands for Numerical Python).

The most fundamental package for performing the scientific computation.

It provides an abundance of useful features for operations on n-arrays and matrices in Python.

The library provides vectorization of mathematical operations on the NumPy array type, which ameliorates performance and accordingly speeds up the execution.



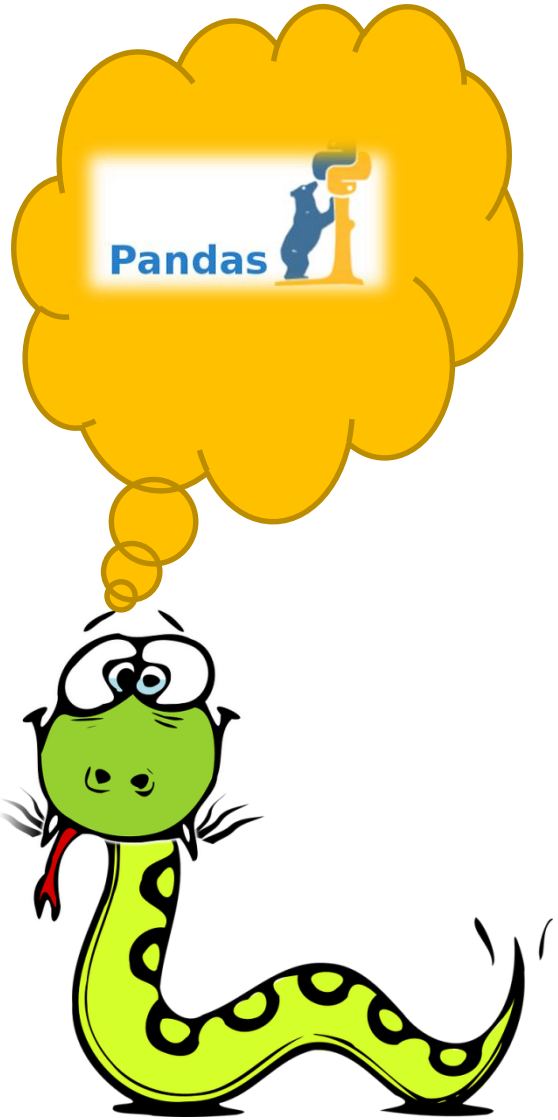
Pandas

Panda (stands for Panel Data).

A perfect tool for data wrangling.

Quick and easy data manipulation, aggregation, and visualization..

A small list of things that you can do with Pandas: Easily delete and add columns from DataFrame, Convert data structures to DataFrame objects, Handle missing data, represents as NaNs, Powerful grouping by functionality



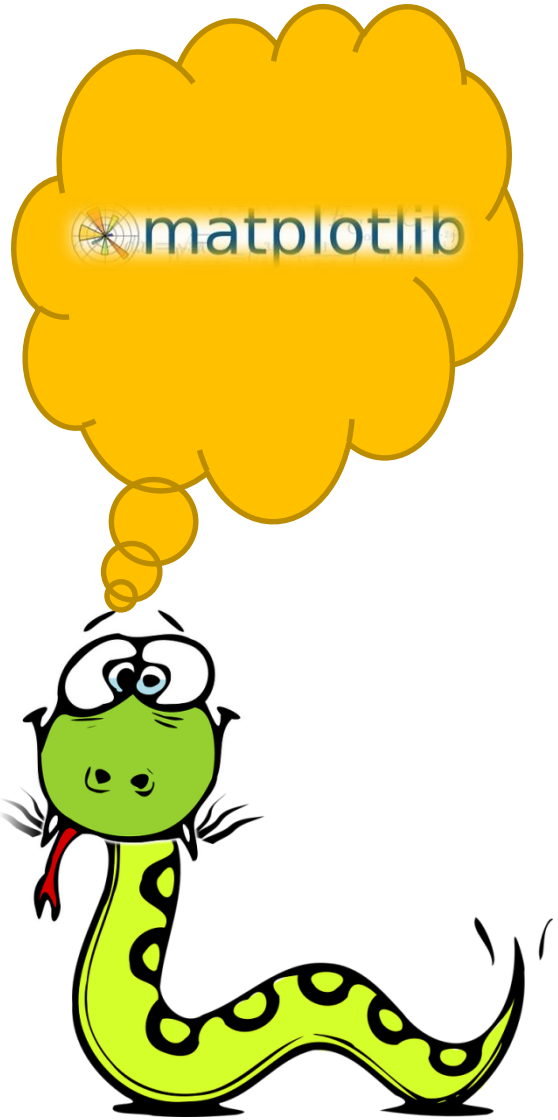
Matplotlib

Matplotlib is the most popular python library for producing plots and other 2D data Visualizations.

facilities for creating labels, grids, legends, and many other formatting entities with Matplotlib. Basically, everything is customizable

Need to write more code to reach the advanced levels of visualizations as compare to MATLAB, or Mathematica.

Make just about any visualizations: Line plots; Scatter plots; Bar charts and Histograms; Pie charts; Stem plots; Contour plots; Quiver plots; Spectrograms.



Scikit-Learn

- Designed for specific functionalities like image processing and machine learning facilitation
- Makes heavy use of its math operations.
- Combines quality code and good documentation, ease of use and high performance.
- It is a de-facto industry standard for machine learning with Python,
  Open source tools for data mining and data analysis

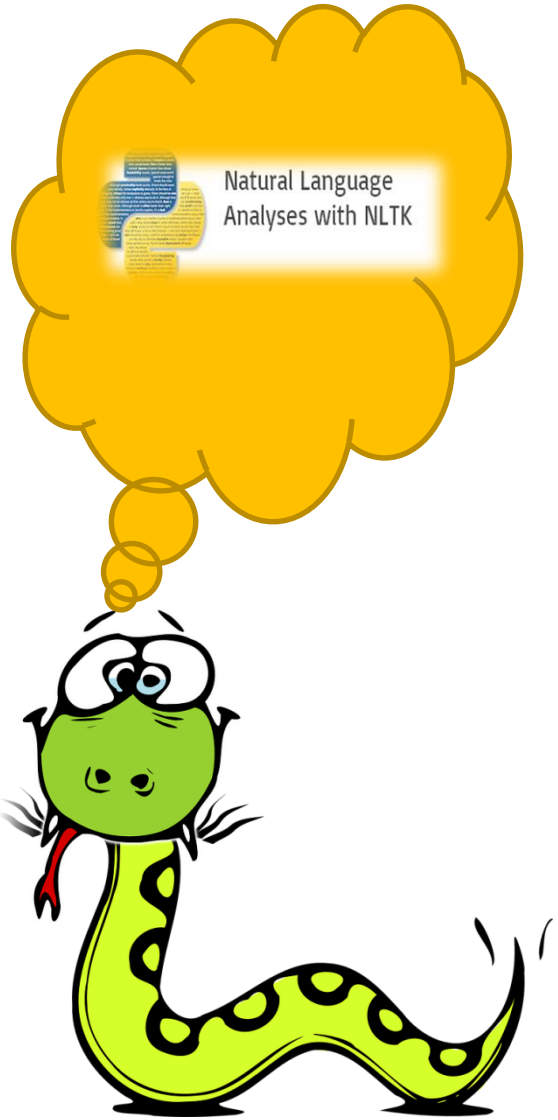
NLTK

NLTK (stands for Natural Language ToolKit).

A leading platform for building Python programs to work with human language data.

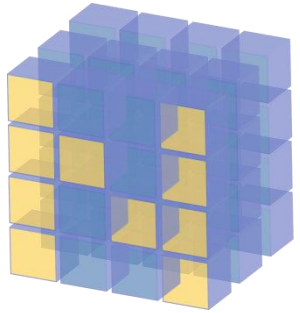
A suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

NLTK also is very easy to learn, actually, it's the easiest natural language processing (NLP) library that you'll use.



Intro to Numpy

proschool
An ims Initiative



NumPy

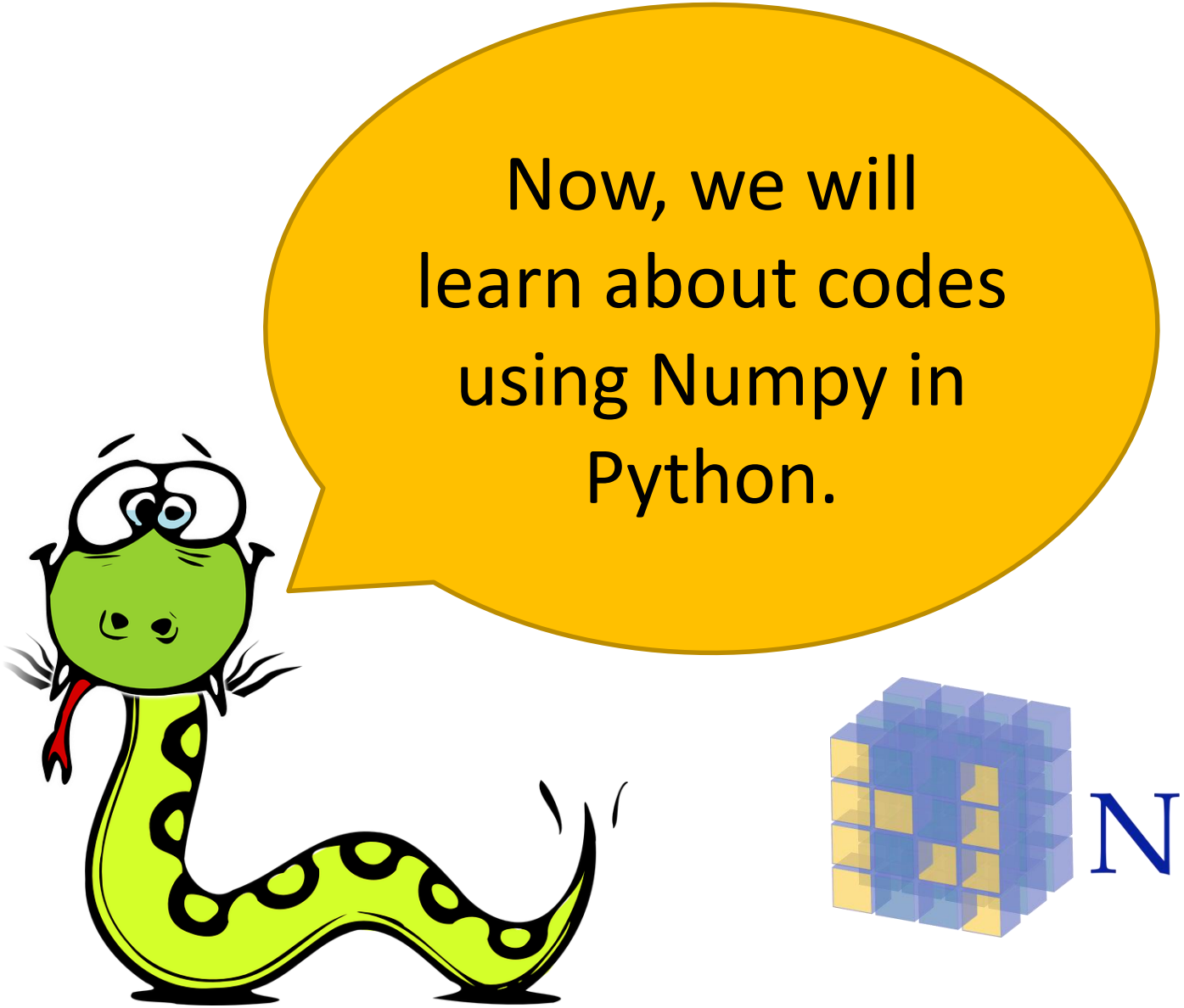
NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Numpy means Numerical Python.

1. NumPy can also be used as an efficient multi-dimensional container of generic data.
2. Arbitrary data-types can be defined.
3. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Numpy

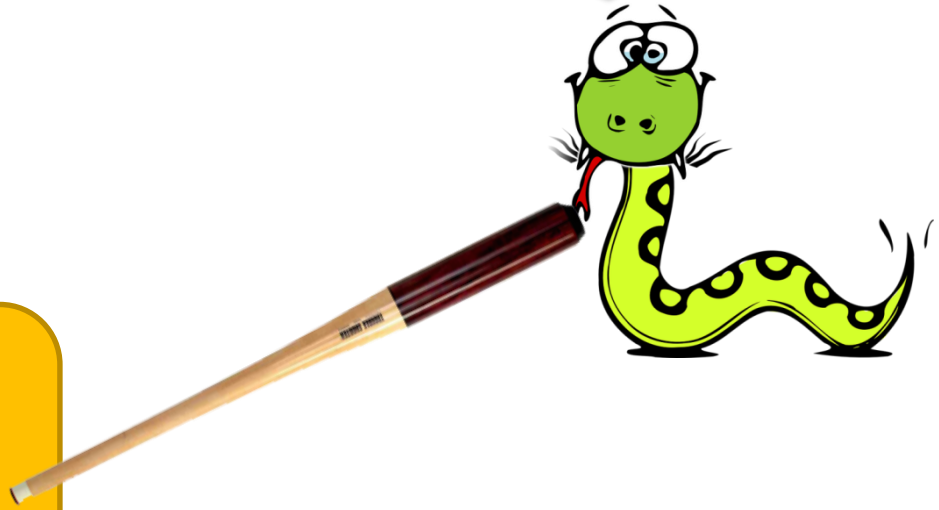


First step to Numpy

IMPORT Numpy Package

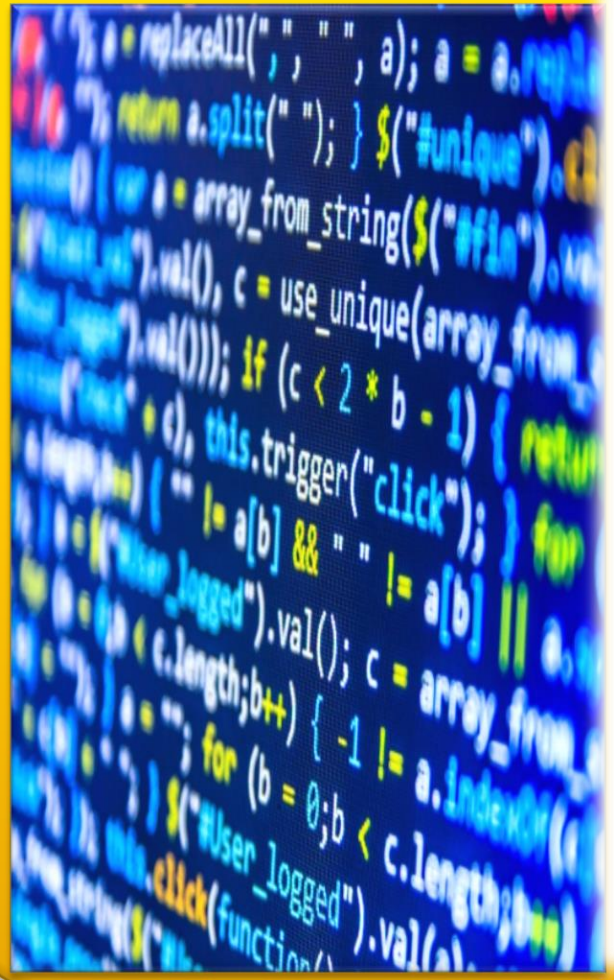
> *Import numpy as np*

Let's play with Numpy and write few codes



```
In [3]: import numpy as np
        a = np.array([1,2,3,4])
        print(a)

        [1 2 3 4]
```

Numpy : list to array



```
In [4]: #define a list
data1 = [[1, 2, 3, 4], [5, 6, 7, 8]]
#create an array from the list
arr1 = np.array(data1)
#find the dimensionality of the array
print("Dimension of array is : ",arr1.ndim)
#find the shape of the array
print("Shape of array is : ",arr1.shape)
#the data type that the array holds
print("Shape of array is : ",arr1.dtype)
#find the class of a data structure
print("Class of data is : ",data1.__class__)
print("Class of array is : ",arr1.__class__)
```

```
Dimension of array is :  2
Shape of array is :  (2, 4)
Shape of array is :  int32
Class of data is :  <class 'list'>
Class of array is :  <class 'numpy.ndarray'>
```

Let's understand

1. How to create array from a list
2. To find different features like dimension, shape and class of numpy array

Numpy : Array to Matrix

proschool
An ims Initiative

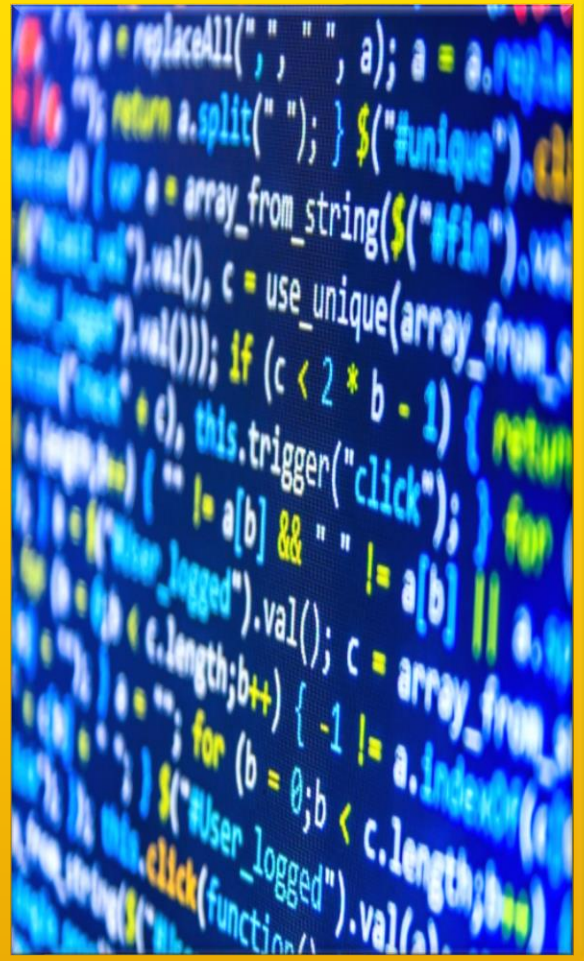
```
In [1]: ar=np.array([1,2,3,4,5,6],float)
        print("the value of ar : ",ar)
        print("the type of ar : ",type(ar))
        nar=ar.reshape(2,3)
        print("Creating ",nar.shape,"matrix using reshape : ",nar)
        far=nar.flatten()
        print("Creating ",far.size,"array using flatten : ",far)

        the value of ar :  [ 1.  2.  3.  4.  5.  6.]
        the type of ar :  <class 'numpy.ndarray'>
        Creating  (2, 3) matrix using reshape :  [[ 1.  2.  3.]
         [ 4.  5.  6.]]
        Creating  6 array using flatten :  [ 1.  2.  3.  4.  5.  6.]
```

Let's use "Reshape " and "Flatten" function to toggle array to matrix and vice -versa

Reshape is used to create matrix of desired shape while , Flatten is used to convert matrix back to array.

Numpy : sequence of integer and float

proschool
An ims Initiative

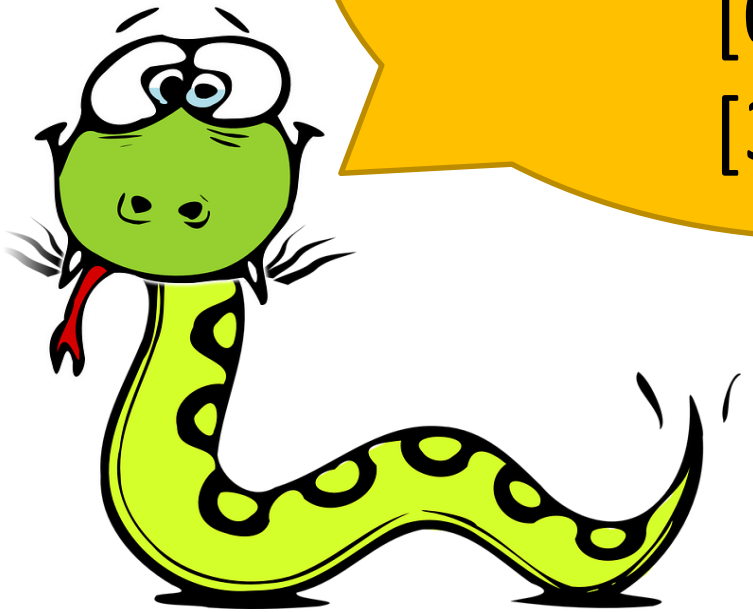


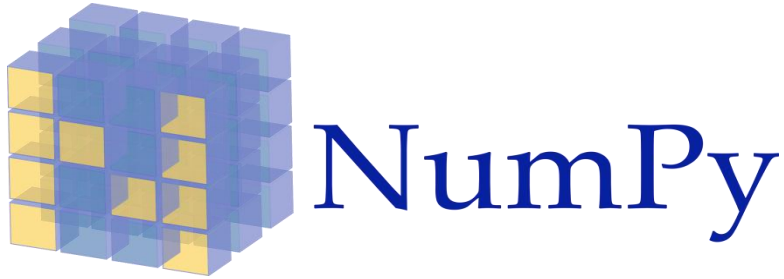
```
In [15]: # sequence of integers with "arange"
         #if one Parameter is been given then sequence starts with 0 till the number
         print("The sequence of 3 numbers from zero : ",np.arange(3))
         # if two Parameters are been given then sequence starts with first parameter
         #till the second parameter
         print("The sequence of 3 numbers from 2 : ",np.arange(2,5))
         # if three Parameters are been given then sequence starts with first parameter
         #till the second parameter with the steps of third parameter
         print("The sequence of 3 numbers from 5 with a difference of 3 : ",np.arange(5,12,3))

         # sequence of integers with "linspace"
         #linspace create sequence with floating numbers, not integers
         #it starts with first parameter, ends with exactly second parameter and steps of
         #third parameter
         print("The sequence of 3 numbers from 5 with a difference of 3 : ",np.linspace(5,11,3))

         #zeroes array/matrix
         print("The array of 5  zeros : ",np.zeros(5))
         print("The matrix of 3 by 5 of zeros : ",np.zeros((3,5)))

The sequence of 3 numbers from zero :  [0 1 2]
The sequence of 3 numbers from 2 :  [2 3 4]
The sequence of 3 numbers from 5 with a difference of 3 :  [ 5  8 11]
The sequence of 3 numbers from 5 with a difference of 3 :  [  5.   8.  11.]
The array of 5  zeros :  [ 0.  0.  0.  0.  0.]
The matrix of 3 by 5 of zeros :  [[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Let's understand

1. How to use "arange"
2. How to use "linspace"
3. How to use "zeros"

Now, a small exercise : create a matrix like this
[0,1,2]
[3,4,5]





Hint : use arange and reshape together

Solution : np.arange(6).reshape(2,3)

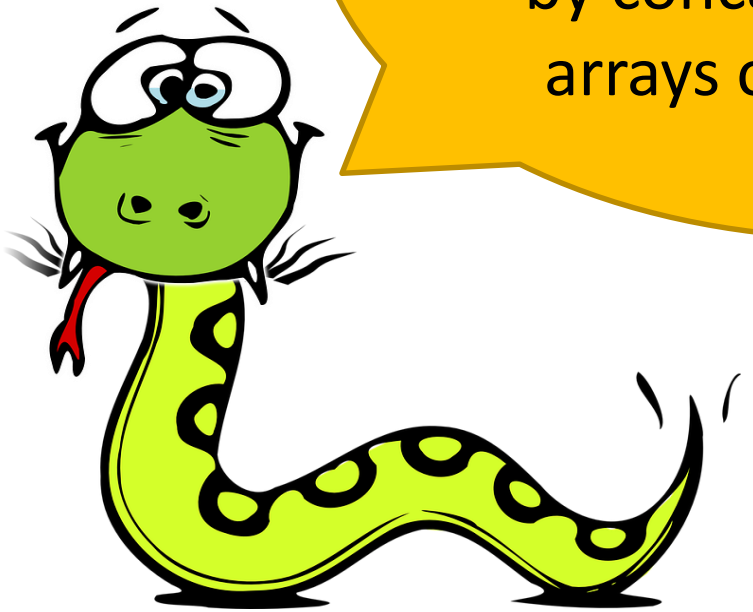
Numpy: Matrix operation

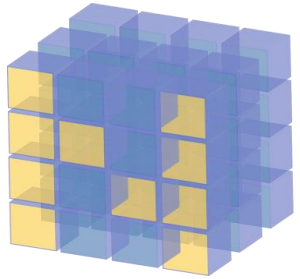
```
In [18]: ar2=np.array([[2,3,6],[3,4,7]],float)
         print("Creating ",ar2.shape,"matrix using numpy : ",ar2)
         far2=ar2.flatten()
         print("Creating ",far2.size,"array using flatten : ",far2)
         ar3=np.unique(ar2)
         print("The unique values of ar2 matrix : ", ar3)
         ar4 = np.transpose(ar2)
         print("The transposed ",ar4.shape, "matrix : ", ar4)
         ar5=np.concatenate((ar3,far2))
         print("the concatenated array is : ",ar5)

Creating  (2, 3) matrix using numpy :  [[ 2.  3.  6.]
 [ 3.  4.  7.]]
Creating  6 array using flatten :  [ 2.  3.  6.  3.  4.  7.]
The unique values of ar2 matrix :  [ 2.  3.  4.  6.  7.]
The transposed  (3, 2) matrix :  [[ 2.  3.]
 [ 3.  4.]
 [ 6.  7.]]
the concatenated array is :  [ 2.  3.  4.  6.  7.  2.  3.  6.  3.  4.  7.]
```

Now you can :

1. Find unique values in a matrix
2. Transpose a matrix
3. Concatenate two or more arrays

Now, a small exercise :
create an array like this
[0,1,2,3,4,5,6,9,11,321]
by concatenating three
arrays of equal length





NumPy

Hint : use concatenate function

Numpy: Solution

```
In [16]: a1=np.array([1,2,3])
         a2=np.array([4,5,6])
         a3=np.array([9,11,321])
         a4=np.concatenate((a1,a2,a3))
         print("the concatenated array is : ",a4)

         the concatenated array is :  [  1   2   3   4   5   6   9  11 321]
```

You are getting better with “numpy” now, isn’t?

Numpy: .tostring function

```
In [5]: import numpy as np
        st=np.array(["h","e","l","l","o"])
        str=st.tostring()
        print("class of st",type(st))
        print("class of str",type(str))
        print("array is : ",st)
        print("string is converted to python bytes : ",str)
```

```
class of st <class 'numpy.ndarray'>
class of str <class 'bytes'>
array is :  ['h' 'e' 'l' 'l' 'o']
string is converted to python bytes :  b'h\x00\x00\x00e\x00\x00\x00l\x00\x00\x00l\x00\x00\x00o\x00\x00\x00
```

Let's understand
How to use convert
array to string

Numpy: Indexing

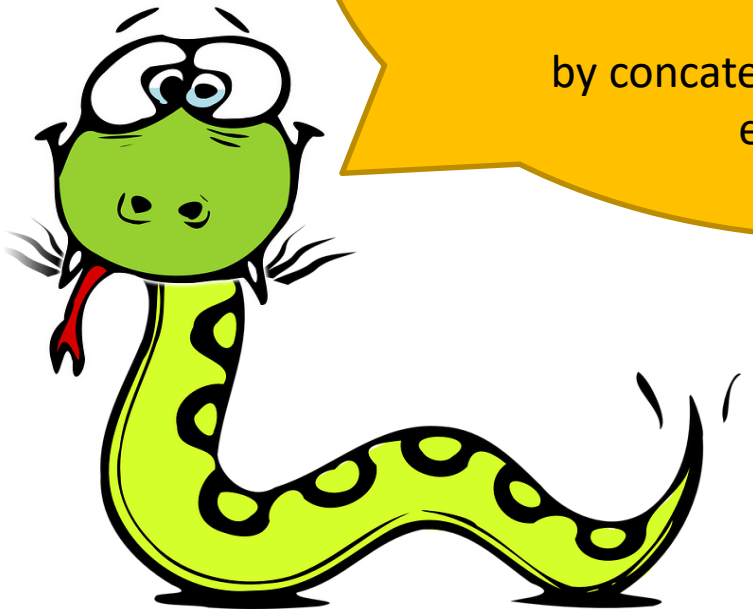
```
In [14]: #indexing
arr=np.arange(15).reshape(3,5)
print("Matrix :\n ",arr)
print("First row of matrix : ",arr[0])
print("First and second rows of matrix :\n ",arr[0:2])
print("Second element of first row of matrix : ",arr[0][1])
print("Third element of second row of matrix : ",arr[1,2])
```

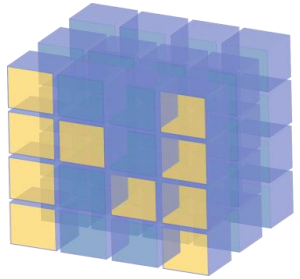
```
Matrix :
  [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
First row of matrix :  [0 1 2 3 4]
First and second rows of matrix :
  [[0 1 2 3 4]
 [5 6 7 8 9]]
Second element of first row of matrix :  1
Third element of second row of matrix :  7
```

Let's understand how to index different element and extract element from matrix/array

Numpy

Now, a small exercise : create a matrix
[1,2]
[4,5]
from arr: [0,1,2]
[3,4,5]
[6,7,8]
by concatenating three arrays of equal length



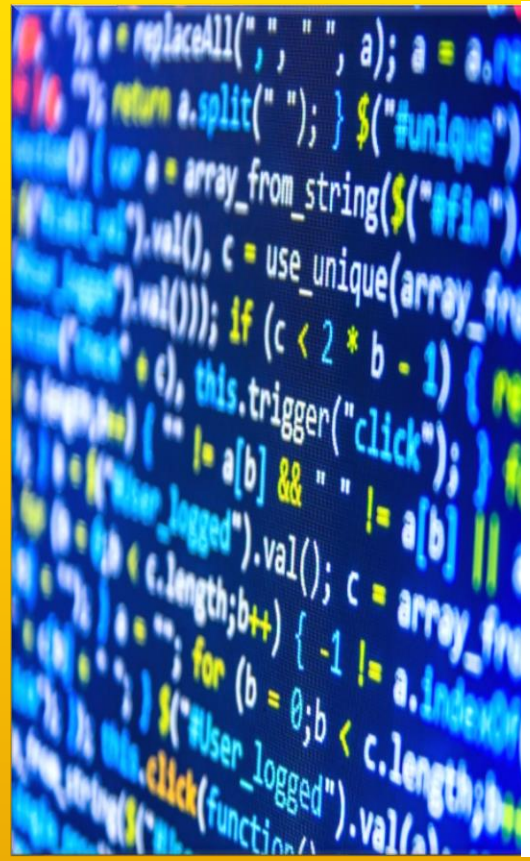


NumPy

Hint : use indexing method

Solution : arr[0:2,1:]

Combining arrays



```
In [34]: arr=np.random.random(10)
         print("Random 10 real numbers :\n ",arr)
         mat=arr.reshape(2,5)
         print("Matrix(2,5) :\n ",mat)
         zer=np.zeros((2,1))
         app=np.append(mat,zer,axis=1)
         print("Matrix after appending (2,6) :\n ",app)
         zer1=np.zeros((1,5))
         app1=np.append(mat,zer1,axis=0)
         print("Matrix after appending (3,5) :\n ",app1)

Random 10 real numbers :
  [ 0.15345285  0.63991069  0.13400976  0.52362345  0.27740633  0.41429625
  0.82621197  0.42737111  0.13313136  0.38874664]
Matrix(2,5) :
  [[ 0.15345285  0.63991069  0.13400976  0.52362345  0.27740633]
 [ 0.41429625  0.82621197  0.42737111  0.13313136  0.38874664]]
Matrix after appending (2,6) :
  [[ 0.15345285  0.63991069  0.13400976  0.52362345  0.27740633  0.        ]
 [ 0.41429625  0.82621197  0.42737111  0.13313136  0.38874664  0.        ]]
Matrix after appending (3,5) :
  [[ 0.15345285  0.63991069  0.13400976  0.52362345  0.27740633]
 [ 0.41429625  0.82621197  0.42737111  0.13313136  0.38874664]
 [ 0.          0.          0.          0.          0.        ]]
```

Let's understand how to create matrix/array with random numbers and to combine them

Mathamatical/Statistical Function

```
In [50]: arr=np.arange(10)
         print("Mean of array : ",arr.mean())
         print("Mean of array : ",np.mean(arr))
         print("Sum of array : ",np.sum(arr))
         print("Standard Deviation of array : ",arr.std())
         print("Square root of array : ",np.sqrt(arr))
         print("Exponential of array : ",np.exp(arr))

Mean of array :  4.5
Mean of array :  4.5
Sum of array :  45
Standard Deviation of array :  2.87228132327
Square root of array :  [ 0.          1.          1.41421356  1.73205081  2.          2.23606798
  2.44948974  2.64575131  2.82842712  3.        ]
Exponential of array :  [  1.00000000e+00   2.71828183e+00   7.38905610e+00   2.00855369e+01
   5.45981500e+01   1.48413159e+02   4.03428793e+02   1.09663316e+03
   2.98095799e+03   8.10308393e+03]
```

Let's understand how to perform different mathematical and statistical task like np.mean() helps to calculate average

There are multiple mathematical and statistical functions with numpy, let's see these functions in detail

Unary functions

Table 4-3. Unary ufuncs

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating point, or complex values. Use `fabs` as a faster alternative for non-complex-valued data
`sqrt`	Compute the square root of each element. Equivalent to `arr ** 0.5`
`square`	Compute the square of each element. Equivalent to `arr ** 2`
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
`ceil`	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
`floor`	Compute the floor of each element, i.e. the largest integer less than or equal to each element
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as separate array
`isnan`	Return boolean array indicating whether each value is NaN (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-NaN) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise. Equivalent to `-arr`.

Here is the list of multiple unary functions with the description. It is applied to individual element.

Misc functions

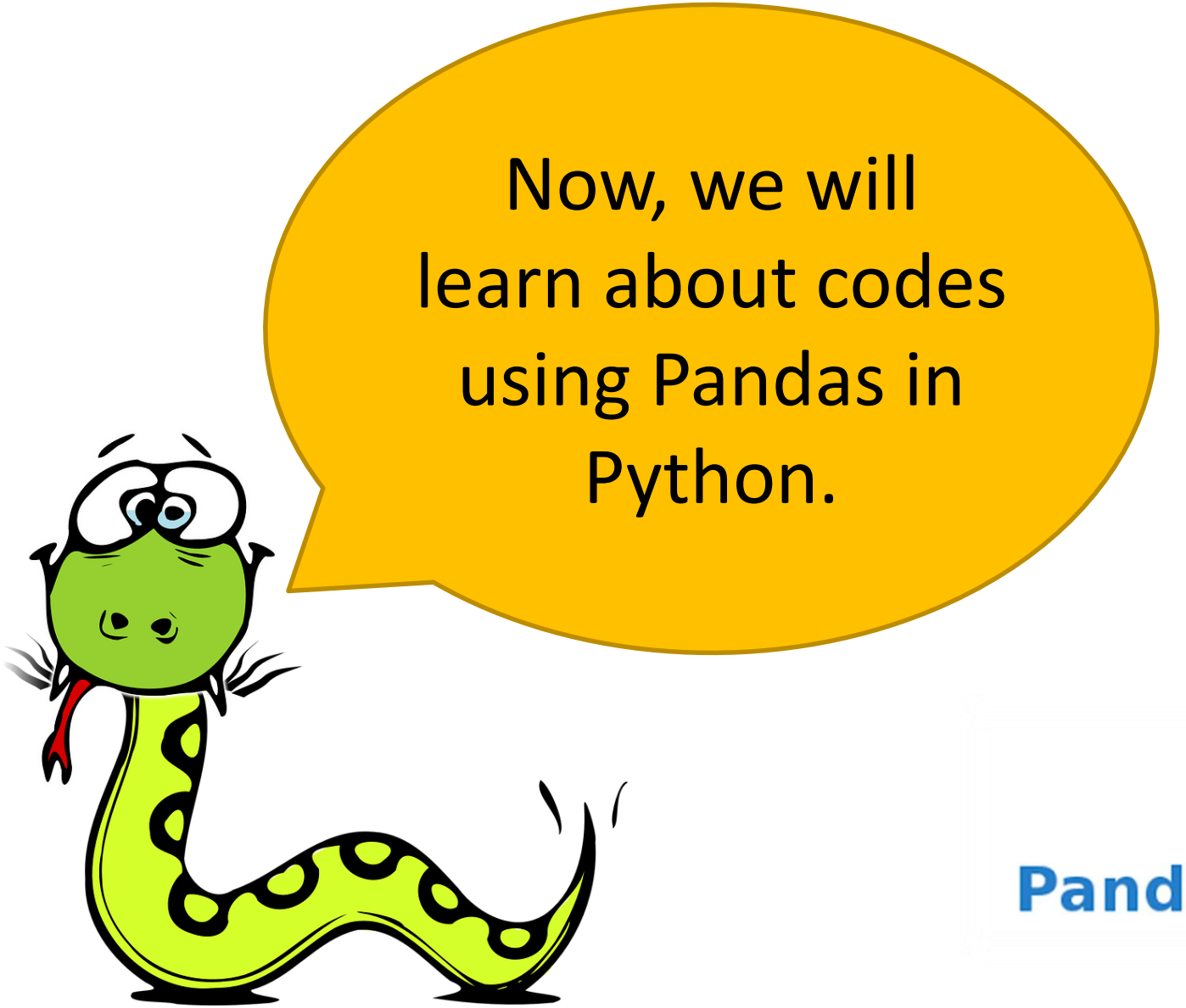
Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum. fmax ignores NaN
minimum, fmin	Element-wise minimum. fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument

Function	Description
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array. Equivalent to infix operators >, >=, <, <=, ==, !=
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation. Equivalent to infix operators &, |, ^

```
In [77]: a1=np.linspace(3,5,3)
         print("first array : ",a1)
         a2=np.linspace(4,4,3)
         print("second array : ",a2)
         print("Using greater_equal function :",np.greater_equal(a1,a2))
         print("Using equal function :",np.equal(a1,a2))

         first array :  [ 3.  4.  5.]
         second array :  [ 4.  4.  4.]
         Using greater_equal function : [False  True  True]
         Using equal function : [False  True False]
```

Some examples of boolean function, to have better understanding. Such functions compare two arrays.

Pandas



Intro to Pandas

What is Pandas? It would be interesting to understand, how does it get its names?

In 2008, developer Wes McKinney started developing **Pandas,** when in need of high performance, flexible tool for analysis of data.
Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word *"Panel Data"* – an Econometrics from Multidimensional data.

Why do we need Pandas?

Pandas is very handy open source library for Data manipulation and analysis.
Prior to Pandas, Python was majorly used for data munging and preparation. It had very less contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data —

- load,
- prepare,
- manipulate,
- model, and
- analyze.

Key Features of Pandas

- Fast and efficient Data Frame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

How to install pandas

- To install through pip
  - *pip install pandas*

- For Windows user
  - If you install Anaconda Python package, Pandas will be installed by default
- For Ubuntu Users
  - *sudo apt-get install python-numpy python-scipy python-matplotlibipythonipythonnotebook*
  - *python-pandas python-sympy python-nose*

Data Structures

- There are three kind of data structures under pandas

Data Structure	Dimension	Definition
Series	1	1D labeled homogeneous array, size immutable.
DataFrame	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array.

- Data structures under Pandas are built on top of Numpy array, which means they are fast.
- All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Examples with codes – Series

Creating a Series	Code Example	Output
Empty Series	s = pd.Series() print (s)	Series([], dtype: float64)
From ndarray	import numpy as np data = np.array(['a','b','c','d']) s = pd.Series(data) print (s)	0 a 1 b 2 c 3 d
From dict	data = {'a' : 0., 'b' : 1., 'c' : 2.} s = pd.Series(data) print (s)	a 0.0 b 1.0 c 2.0
From scalar	s = pd.Series(5, index=[0, 1, 2, 3]) print (s)	0 5 1 5 2 5 3 5

Examples with codes – DataFrames creation

proschool
An ims Initiative

Creating a DataFrame	Code Example	Output
Empty DataFrames	df = pd.DataFrame() print (df)	Empty DataFrame Columns: [] Index: []
From list	data =[['Alex',10],['Bob',12],['Clarke',13]] df=pd.DataFrame(data,columns=['Name','Age']) print (df)	Name Age Alex 10 Bob 12 Clarke 13
From dict	d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])} df = pd.DataFrame(d) print (df)	one two a 1.0 1 b 2.0 2 c 3.0 3 d NaN 4

DataFrames Column operation

proschool
An ims Initiative

DataFrame	Code Example	Output
Column Selection	d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])} df = pd.DataFrame(d) print (df ['one'])	a 1.0 b 2.0 c 3.0 d NaN Name: one, dtype: float64
Column Addition	df['three']=pd.Series([10,20,30],index=['a','b','c']) print (df)	one two three a 1.0 1 10.0 b 2.0 2 20.0 c 3.0 3 30.0 d NaN 4 NaN
Column Deletion	del df['one'] print (df)	two three a 1 10.0 b 2 20.0 c 3 30.0 d 4 NaN

DataFrames Row operation

proschool
An ims Initiative

DataFrame	Code Example	Output
Selection by Label	d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])} df = pd.DataFrame(d) print (df.**loc**['b'])	one 2.0 two 2.0 Name: b, dtype: float64
Selection by Integer	df = pd.DataFrame(d) print (df.**iloc**[2])	one 3.0 two 3.0
Slice Rows	df = pd.DataFrame(d) print (df[2:4])	one two c 3.0 3 d NaN 4
Addition rows	df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b']) df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b']) df = df.append(df2) print (df)	a b 0 1 2 1 3 4 0 5 6 1 7 8

Deletion Rows	# Drop rows with label 0
	df = df.drop(0)
	print (df)
		a b
		1 3 4
		1 7 8

Panel - Introduction

- A **panel** is a 3D container of data. The term **Panel data** is derived from econometrics and is partially responsible for the name pandas − **pan(el)-da(ta)**-s.
- The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data. They are −
  - **items** − axis 0, each ***item*** corresponds to a DataFrame contained inside.
  - **major_axis** − axis 1, it is the index (***rows***) of each of the DataFrames.
  - **minor_axis** − axis 2, it is the ***columns*** of each of the DataFrames.

Examples with codes – Panel creation

proschool
An ims Initiative

To create Panda- Code is mentioned below

pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)

Panel	Code Example	Output
Create Empty Panel	import pandas as pd import numpy as np data = np.random.rand(2,4,5) p = pd.Panel(data) print (p)	<class 'pandas.core.panel.Panel'> Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis) Items axis: 0 to 1 Major_axis axis: 0 to 3 Minor_axis axis: 0 to 4
Selection of data from panel	data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)), 'Item2' : pd.DataFrame(np.random.randn(4, 2))} p = pd.Panel(data) print (p['Item1'])	0 1 2 0 0.488 -0.128 0.930 1 0.417 0.896 0.576 2 -2.775 0.571 0.290 3 -0.400 -0.144 1.110

Basic Functions

S.No.	Attribute or Method	Description
1	axes	Returns a list of the row axis labels.
2	dtype	Returns the datatype of the object.
3	empty	Returns True if series is empty.
4	ndim	Returns the number of dimensions of the underlying data, by definition 1.
5	size	Returns the number of elements in the underlying data.
6	values	Returns the Series as ndarray.
7	head()	Returns the first n rows.
8	tail()	Returns the last n rows.

Examples – Basic Function using Pandas

proschool
An ims Initiative

Function	Code Example	Output
ndim	s = pd.Series(np.random.randn(3)) print (s) print ("The dimensions of the object:") print (s.ndim)	0 0.175898 1 0.166197 2 -0.609712 dtype: float64 The dimensions of the object: 1
size	s = pd.Series(np.random.randn(2)) print s print ("The size of the object:") print (s.size)	0 3.078058 1 -1.207803 dtype: float64 The size of the object: 2
head	s = pd.Series(np.random.randn(4)) print ("The original series is:") print (s) print ("The first two rows :") print (s.head(2))	The original series is: 0 0.720876 1 -0.765898 2 0.479221 3 -0.139547 dtype: float64 The first two rows : 0 0.720876 1 -0.765898 dtype: float64

Examples – contd..

proschool
An ims Initiative

Function	Code Example	Output
dtypes- Returns the data type of each column	d={'Name':pd.Series(['Tom','James']), 'Age':pd.Series([25,26]), 'Rating':pd.Series([4.23,3.24])} df = pd.DataFrame(d) print ("The data types are:") print (df.dtypes)	The data types are: Age int64 Name object Rating float64 dtype: object
Transpose (T)	d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']), 'Age':pd.Series([25,26,25,23,30,29,23]), 'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])} df = pd.DataFrame(d) print ("The transpose of the data series is:") print (df.T)	The transpose of the data series is: 0 1 2 3 4 5 6 Age 25 26 25 23 30 29 23 Name Tom James Ricky Vin Steve Smith Jack Rating 4.23 3.24 3.98 2.56 3.2 4.6 3.8

Descriptive Analytics

S.No.	Attribute or Method	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()/max()	Minimum/Maximum Value
8	abs()	Absolute Value
9	prod()	Product of Values
10	cumsum()	Cumulative Sum
11	cumprod()	Cumulative Product
12	describe()	Summary of statistics

Examples – Descriptive Analytics

proschool
An ims Initiative

Descriptive	Code Example	Output
sum()	d={'Name':pd.Series(['Tom','James']), 'Age':pd.Series([25,26]), 'Rating':pd.Series([4.5,3.5])} df = pd.DataFrame(d) print (df.sum())	Name TomJames Age 51 Rating 8 dtype: object
mean()	d={'Name':pd.Series(['Tom','James']), 'Age':pd.Series([25,26]), 'Rating':pd.Series([4.5,3.5])} df = pd.DataFrame(d) print (df.mean())	Age 25.5 Rating 4.0 dtype: float64
describe()	d={'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David ','Gasper','Betina','Andres']), 'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]), 'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])} pd.DataFrame(d) print (df.describe())	Age Rating count 12 12 mean 31 3 std 9 0.66 min 23 2.56 25% 25 3.23 50% 29.5 3.79 75% 35.5 4.13 max 51 4.8

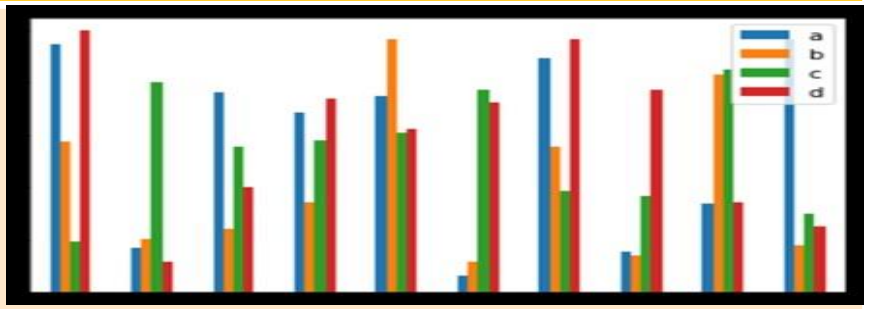
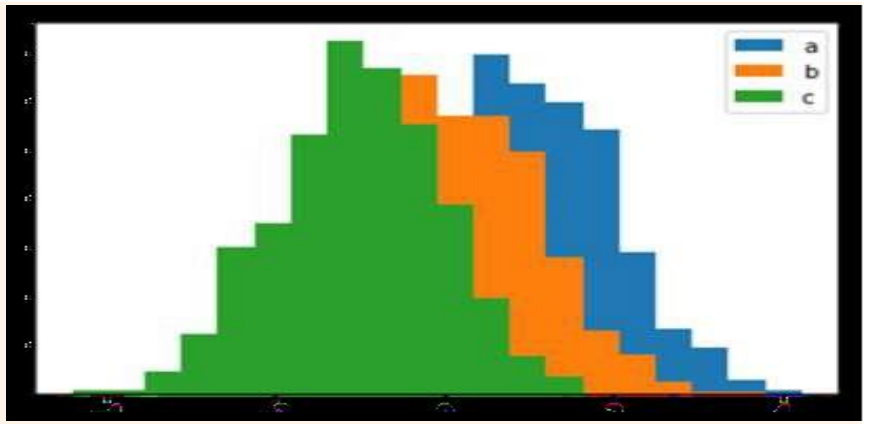
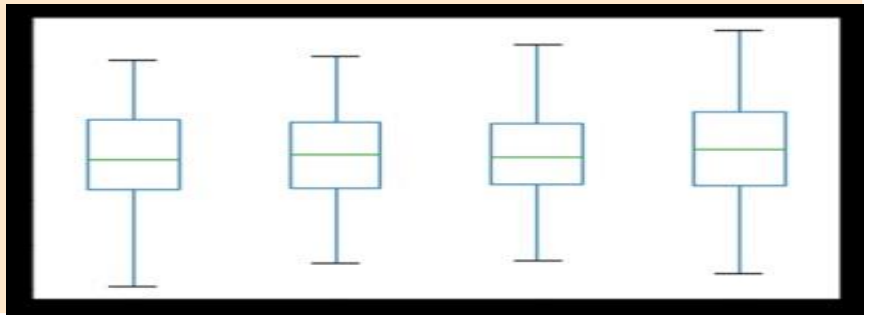
I/O operations

The two workhorse functions for reading text files (or the flat files) are
read_csv()
read_table().
They both use the same parsing code to intelligently convert tabular data into a DataFrame object −

pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None

I/O operation	Code Example	Output
Read.csv	import pandas as pd df=pd.read_csv("temp.csv") print (df)	S.No Name Age City Salary 0 1 Tom 28 Toronto 20000 1 2 Lee 32 HongKong 3000 2 3 Steven 43 Bay Area 8300 3 4 Ram 38 Hyderabad 3900
Skiprows	import pandas as pd df=pd.read_csv("temp.csv", skiprows=2) print (df)	2 Lee 32 HongKong 3000 0 3 Steven 43 Bay Area 8300 1 4 Ram 38 Hyderabad 3900

Visualisation

Plots	Code Example	Output
Bar Plot	import pandas as pd import numpy as np df=pd.DataFrame(np.random.rand(10,4),columns=['a', 'b','c','d') df.plot.bar()	
Histogram	import pandas as pd import numpy as np df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c']) df.plot.hist(bins=20)	
Box Plot	import pandas as pd import numpy as np df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E']) df.plot.box()	

Merge dataframes

proschool
An ims Initiative

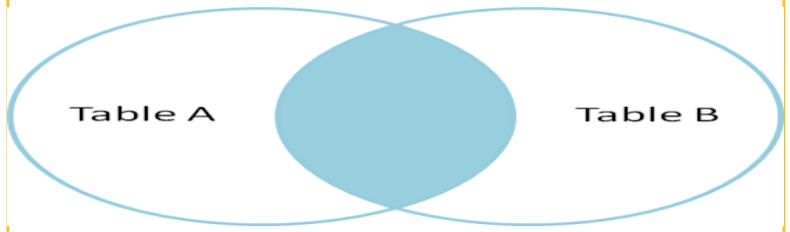
SYNTAX
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True)

Merge	Code Example	Output
Merge on two dataframe	import pandas as pd left = pd.DataFrame({ 'id':[1,2,3,4,5], 'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],'subject_id':['sub1','sub2','sub4','sub6','sub5']}) right = pd.DataFrame({'id':[1,2,3,4,5], 'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],'subject_id':['sub2','sub4','sub3','sub6','sub5']}) print(pd.merge(left,right,on='id'))	Name_x id subject_id_x Name_y subject_id_y 0 Alex 1 sub1 Billy sub2 1 Amy 2 sub2 Brian sub4 2 Allen 3 sub4 Bran sub3 3 Alice 4 sub6 Bryce sub6 4 Ayoung 5 sub5 Betty sub5

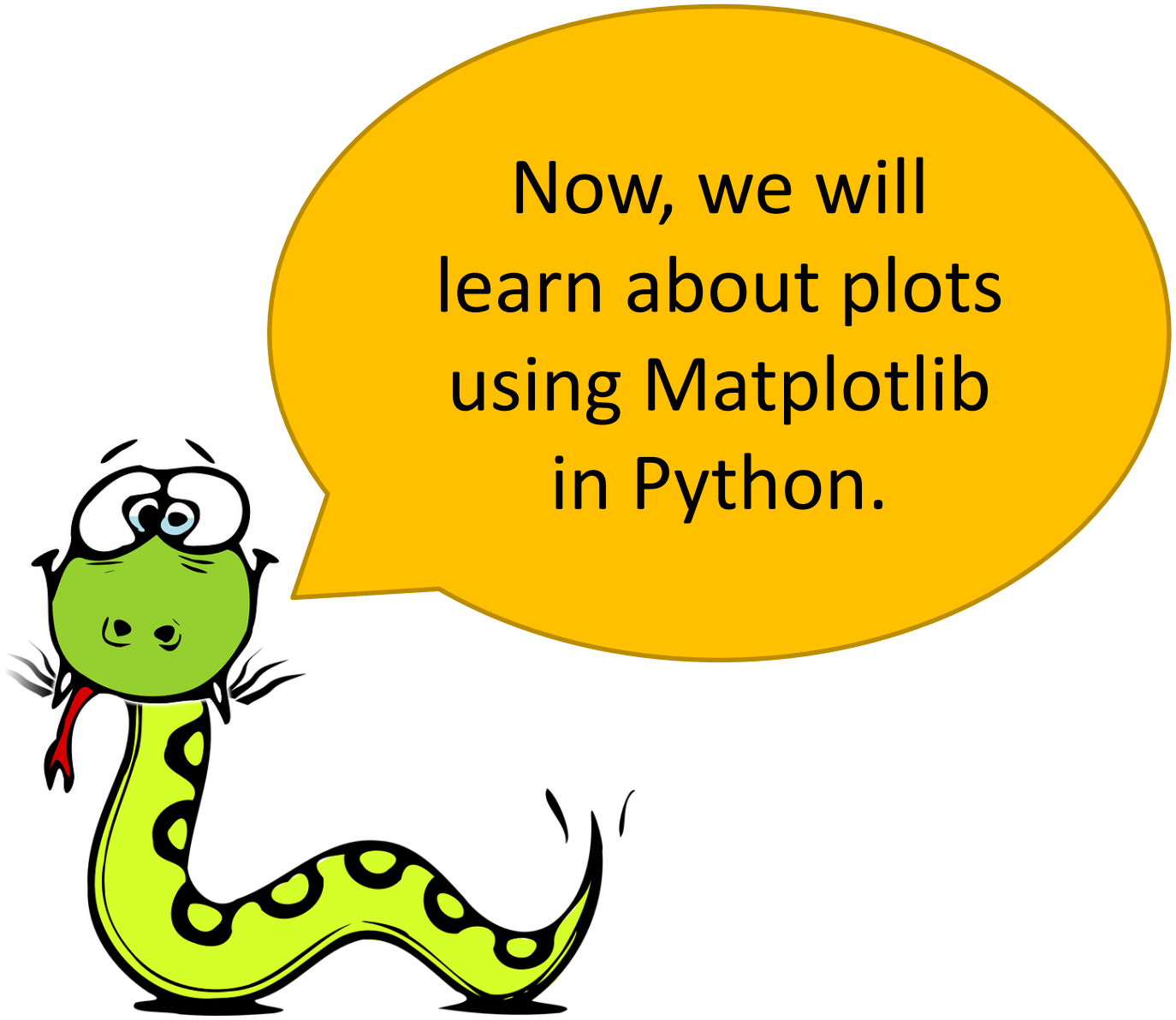
Example contd..

Merge	Code Example	Output
Using two Id keys	print (pd.merge(left,right,on=['id','subject_id']))	Name_x id subject_id Name_y 0 Alice 4 sub6 Bryce 1 Ayoung 5 sub5 Betty
Inner Join using “How”	print (pd.merge(left, right, on='subject_id', how='inner'))	Name_x id_x subject_id Name_y id_y 0 Amy 2 sub2 Billy 1 1 Allen 3 sub4 Brian 2 2 Alice 4 sub6 Bryce 4 3 Ayoung 5 sub5 Betty 5

Joins summarised

Merge Method	Code	Description
left	pd.merge(left, right, on='subject_id', how='left')	
right	pd.merge(left, right, on='subject_id', how='right')	
outer	pd.merge(left, right, on='subject_id', how='outer')	
inner	pd.merge(left, right, on='subject_id', how='inner')	

Matplot



Matplotlib

- Matplotlib is very useful plotting library in python. We can produce 2D plots using matplotlib library.
- It provides a very easy way to visualize data and the quality of figure produced are publishable.
- Matplotlib has a very large number of object-oriented API support, by the help of which we can embed the graphics/plots produced by matplotlib into GUI of software applications.
- Loading matplotlib: We can load the matplot lib library by using import statement. i.e. **import matplotlib**

Types of Plots

- In this tutorial we are mainly focusing on following 5 types of plots
  1. Bar Plot
  2. Stacked Bar Plot
  3. Histogram
  4. Line Chart
  5. Box plot

- In subsequent slides We are going to explain each mentioned plots with relevant examples:

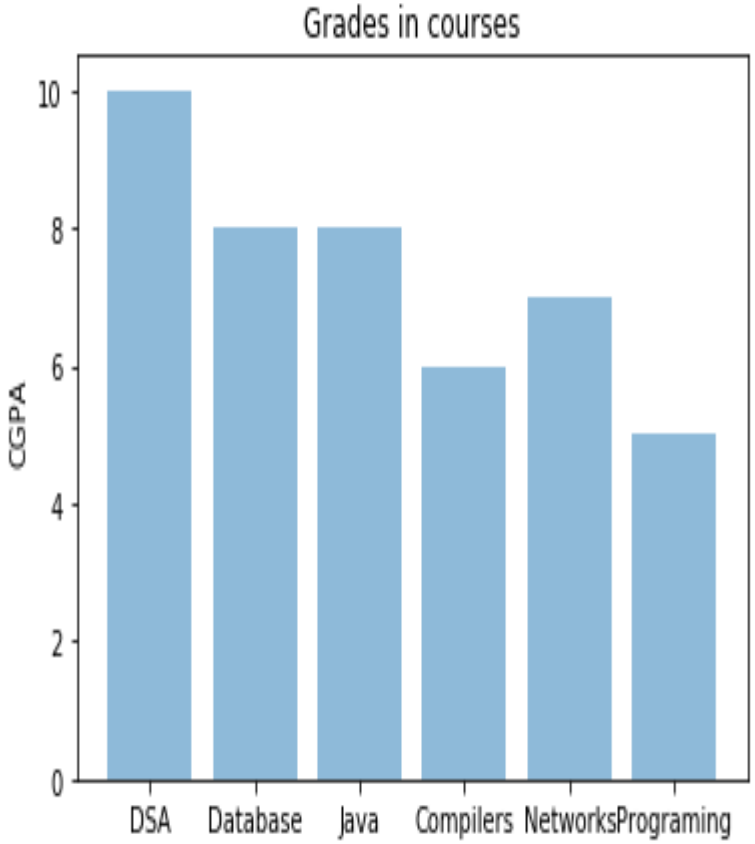
Bar Charts

- Bar Charts or bar graphs are used to present grouped data with rectangular bars
- The lengths of the bar are proportional to the values that they represent.
- The bars can be plotted vertically or horizontally. A vertical bar chart is sometimes called a Line graph.
- There are two axis to the bar plots:
  - **First Axis**: It includes categorical values(Qualitative).
  - **Second Axis**: It includes discrete values(Quantitative)
- Example:
  - Consider an example of course wise average CGPA of a class.
  - In this case name of courses(Categorical) becomes X-axis and CGPA(Discrete) becomes y-axis.

Bar Chart Example

proschool
An ims Initiative

```
# This is a simple example of Bar chart where we have given y axis and x-axis
# and matplotlib is used to draw the bar graph

courses = ('DSA', 'Database', 'Java', 'Compilers', 'Networks', 'Programing')
y_pos = np.arange(len(courses))
# y_pos = [0, 1, 2, 3, 4, 5]
cgpa = [10, 8, 8, 6, 7, 5]
# axis() is not mandatory in any of the charts, if we dont give axis, matplotlib will automatically
# set the axis according to inputs
plt.bar(y_pos, cgpa, align='center', alpha=0.5)
# xticks attach courses names with cgpa
plt.xticks(y_pos, courses)
plt.ylabel('CGPA')
plt.title('Grades in courses')
plt.show()
```

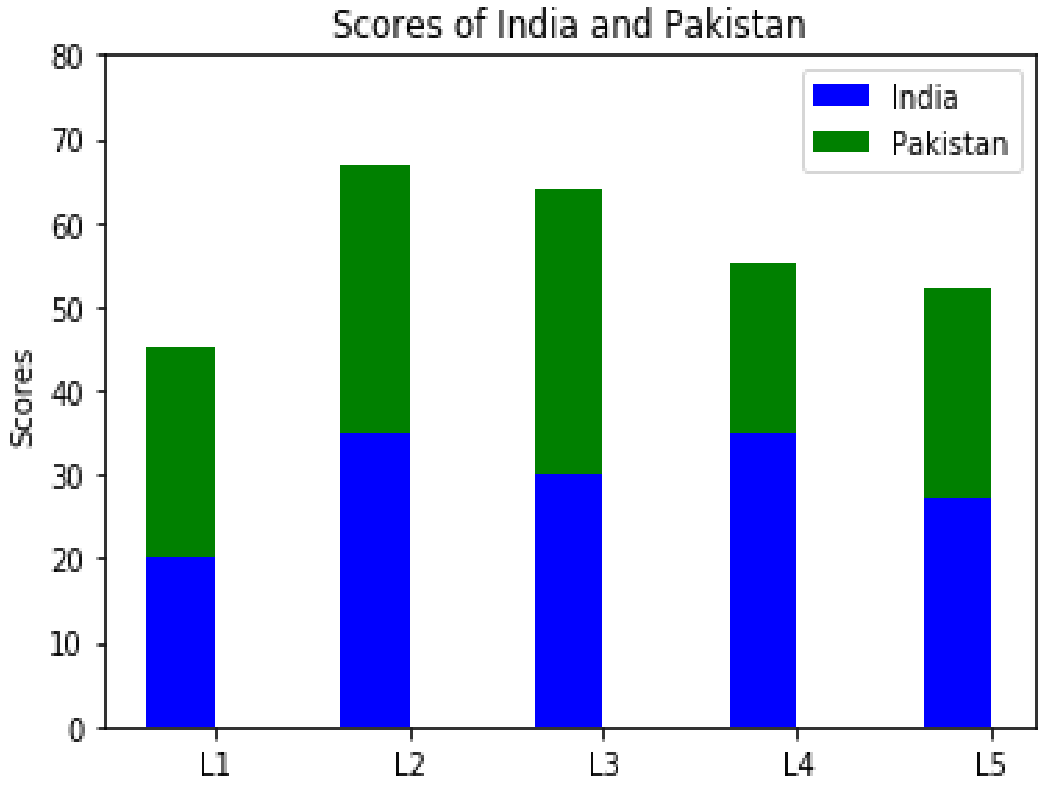


Stacked Bar Plot

- Stacked bar plot is nothing but a bar plot which used to plot similar categories with different discreet value combined together.
- Different discrete values are represented as different colors in one column.
- Example:
  - If you are comparing scores in final 5 overs of India and Pakistan in previous 5 matches then stacked bar plot gives better comparison.
  - Both the teams are represented with different colors.
  - India – Represented as Blue color and Pakistan – Represented as Green color

Stacked Bar Chart Example

```
# Here we are showing india and pakistan scores in last 5 overs
india = (20, 35, 30, 35, 27)
pakistan = (25, 32, 34, 20, 25)
N = 5
ind = np.arange(N)    # the x locations for the groups
width = 0.35       # the width of the bars: can also be len(x) sequence
# This creates bars of both teams, in one graph
p1 = plt.bar(ind, india, width, color='b')
p2 = plt.bar(ind, pakistan, width, color='g', bottom=india)


plt.ylabel('Scores')
plt.title('Scores of India and Pakistan')
# Shows the text between the center of each bar
plt.xticks(ind + width/2., ('L1', 'L2', 'L3', 'L4', 'L5'))
plt.yticks(np.arange(0, 81, 10))


# shows the box labeling of india and pakistan
plt.legend((p1[0], p2[0]), ('India', 'Pakistan'))
plt.show()
```

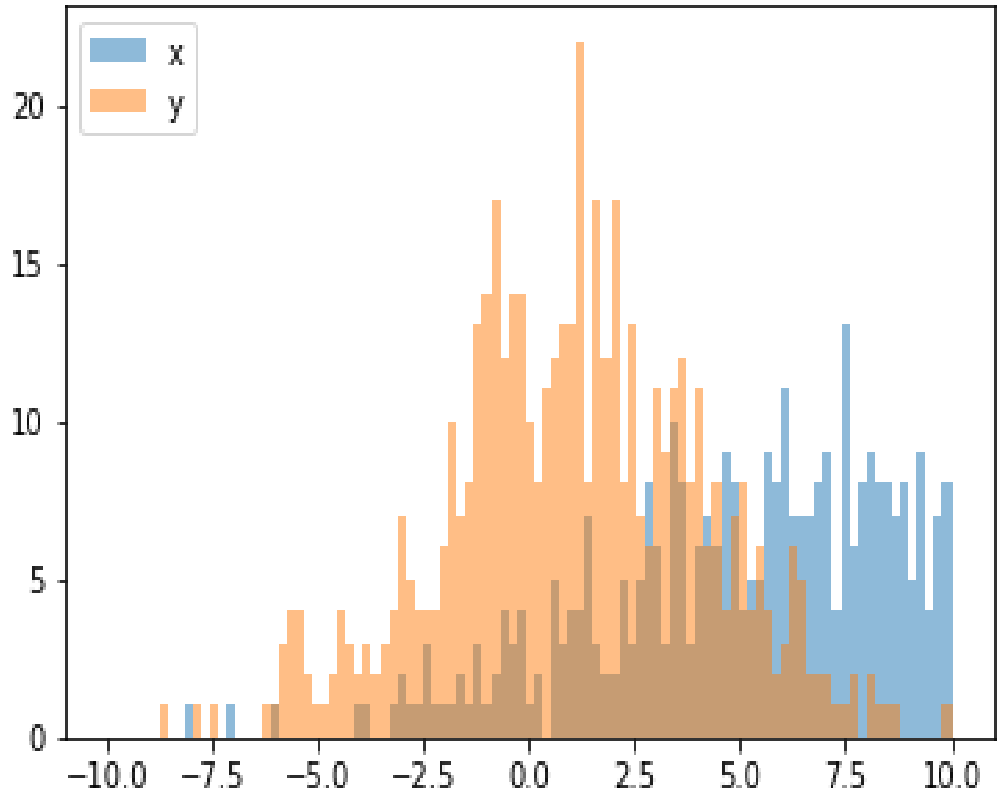


Histogram

- Histogram is used for density estimation
- It uses categorical data similar to bar plot, but groups continuous data into numbers into ranges.
- The ranges of continuous data are known as bins/Intervals.
- One more important point is all the bins must be adjacent to each other.
- Usually bins are of equal size, but they can vary in sizes.
- Example:
  - Consider a histogram of **x** and **y**, which are 500 Gaussian generated random numbers. In the histogram distribution of **x** and **y** are represented with different colors.

Histogram

```
# Histogram
# Here is an example of histogram using matplotlib
x = [random.gauss(8,5) for _ in range(500)]
y = [random.gauss(1,3) for _ in range(500)]
# Here x and y are 500 gaussian generated random numbers

# bin shows the x axis and next variable is number of bars
# Here x axis is from -10 to 10 and 100 is the number of bars in histogram
bin = np.linspace(-10, 10, 100)

# hist() is used to build the histogram
# alpha is the brightness of the color of the histogram
plt.hist(x, bin, alpha=0.5, label='x')
plt.hist(y, bin, alpha=0.5, label='y')

# legend is the postion of labelled box
plt.legend(loc='upper left')
plt.show()
```

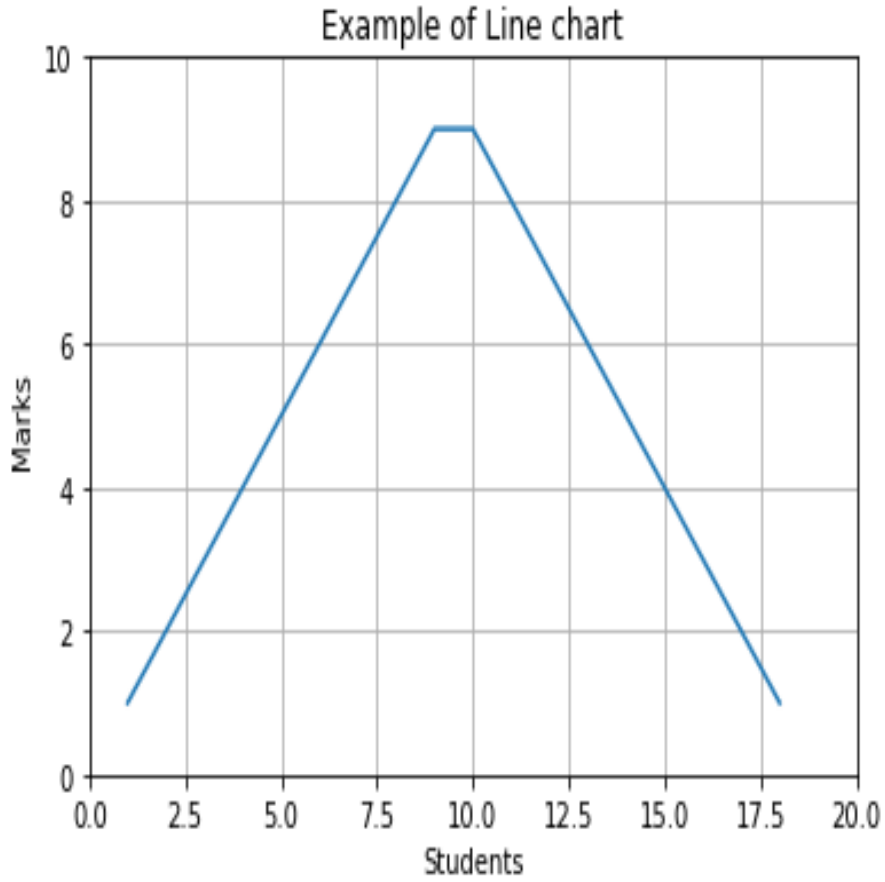


Line Plot

- Used to visualize a trend in data over intervals of time
- A line chart or line graph is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments.
- **Example:**
  - To identify the trend in the results of students we plot a line chart.
  - X-axis – Number of students ranging from 0 to 20
  - Y-axis – Obtained marks/CGPA ranging from 0 to 10

Line Chart

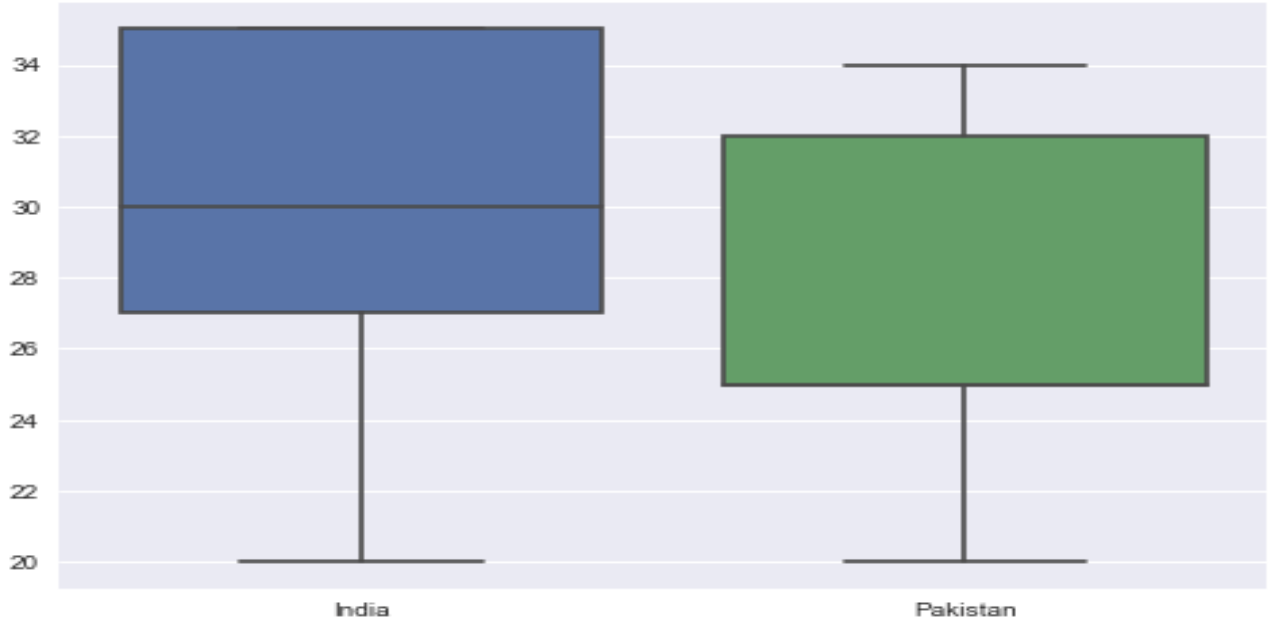

```
# Line chart

y = np.array([1,2,3,4,5,6,7,8,9,9,8,7,6,5,4,3,2,1])
x =[]
for i in range (1,19):
    x.append(i)

# Line chart can be drawn using plot() function
plt.plot(x,y)

# axis() is used to set the range of x axis and y axis
# Here I have set x-axis from 0-20 and y axis from 0-10
plt.axis([0,20,0,10])

# xlabel and ylabel is used to set the label of x-axis and y-axis
plt.xlabel('Students')
plt.ylabel('Marks')

plt.title('Example of Line chart')
plt.grid(True)

# To save the graph as a image, uncomment the following line
# plt.savefig("cgp.png")
plt.show()
```



Boxplot

- We draw boxplot to show the distribution with respect to categories.
- Usually boxplots give the good comparison between categories.
- Box represents different quartiles and viscous extends to show the rest of distribution.
- Boxplots are very helpful to “outlier” detection.
- Example: Let us see the distribution of India and Pakistan’s final 5 over scores in last 5 matches.

```
In [97]: Scores = [20, 35, 30, 35, 27, 25, 32, 34, 20, 25]
    ...: Country = ['India', 'India', 'India', 'India', 'India',
'Pakistan', 'Pakistan', 'Pakistan', 'Pakistan', 'Pakistan']
    ...: sns.boxplot(x=Country,y=Scores)
```

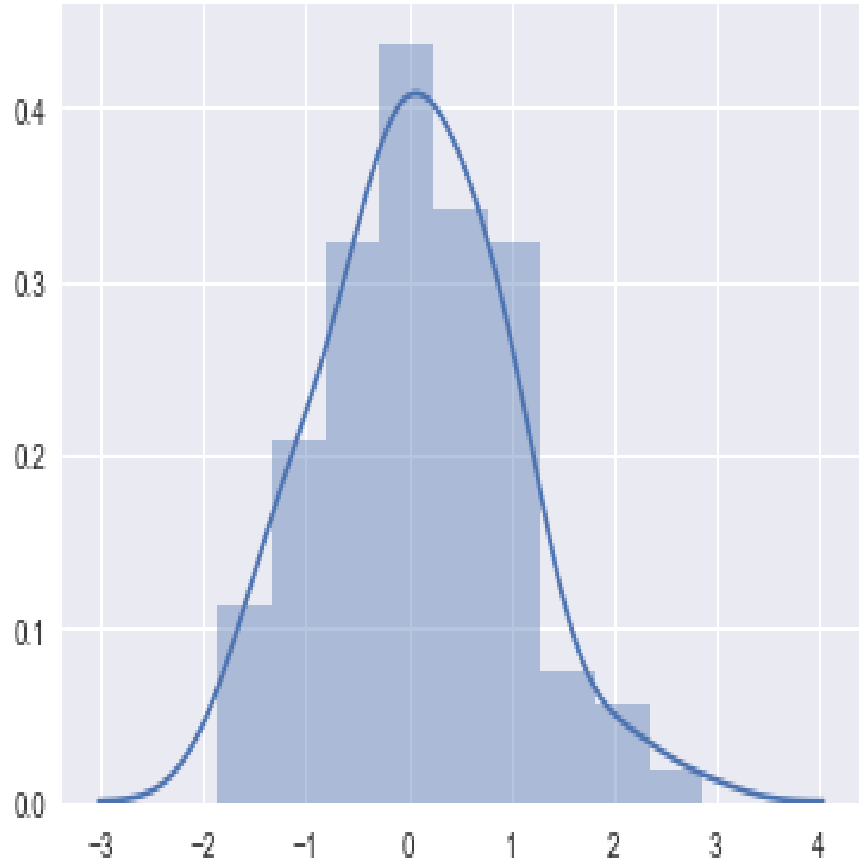


Univariate Distributions

- Probability distribution of single random variable is known as univariate distribution.
- We can have a quick look on univariate distribution by one of python's library Seaborn has a function distplot(),which draws a histogram and fit a kernel density estimate (KDE) of the input data.
- **Example:**
  - Consider a variable x is a list of 100 random samples from a Gaussian distribution. We can plot this random variable x by using sns.distplot()

Distplot Example


```
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "distributions")))

# x is a list of 100 random samples from a Gaussian distribution
x = np.random.normal(size=100)

# sns.displot() draws a histogram and fit a kernel density estimate (KDE) of the input data
# if we set hist = true , it will show histogram, with rug = true it will show KDE
#sns.distplot(x, hist=True, rug=True);

# Also  uncomment each of the following statements one by one, but first remove the above line
sns.distplot(x, hist=True, rug=False);
#sns.distplot(x, hist=False, rug=True);

plt.show()
```



Plots on imported file

- In coming slides let us discuss about some basic graphs/plots used in matplotlib with the help of **mtcars** dataset.

Loading data to plot


```
#Load data
dat = pd.read_csv("E:\IMS\mtcars.csv", header=0)
dat.head()
```

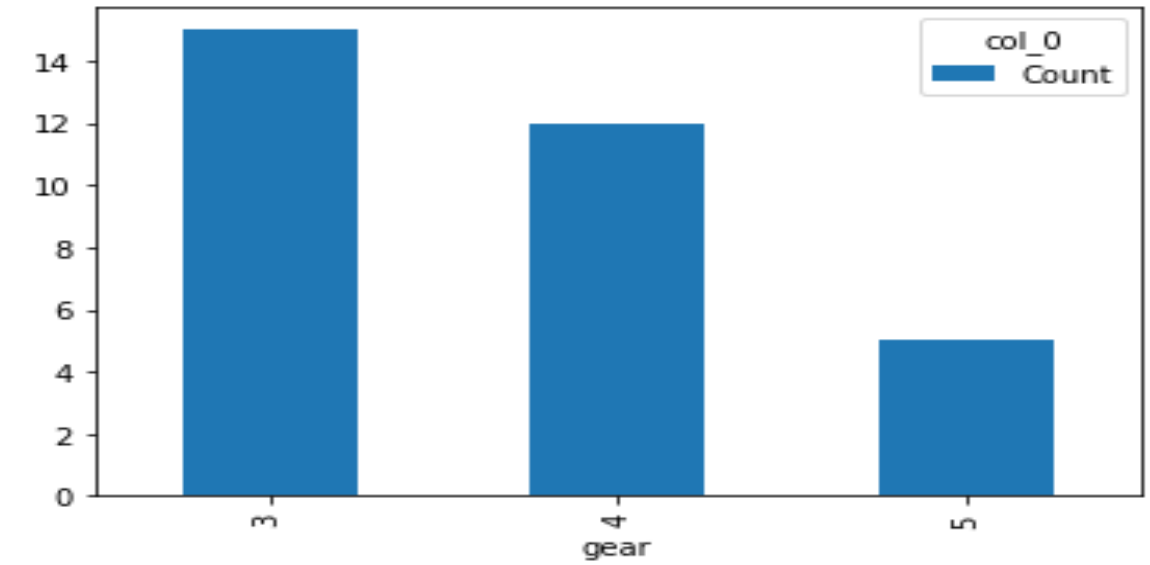
	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Bar Plot

- **Bar plot**: Let us plot the bar plots for some categorical variables in mtcars dataset.
- **Normal bar plot**: Here we are plotting the bar plot of categorical variable **"gear"** from **mtcars** dataset. Using **pd.crosstab(x,y)** we are giving axises and using **plot()** we are plotting the graph, here we need bar plot so **"kind = bar"**. In below plot x-axis contains possible gear numbers of vehicles are taken and in y-axis count of vehicles with respect to their gears is taken.

```
pd.crosstab(dat['gear'],columns='Count').plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc1cc668>
```

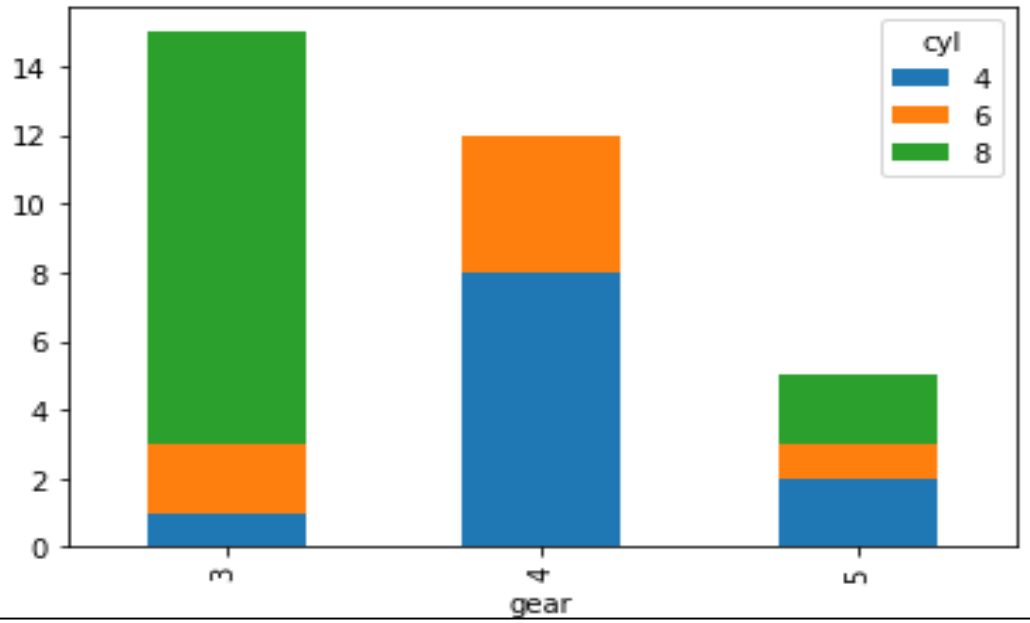


Stacked Bar Plot

- **Vertical Stacked Bar Plot**: We are visualizing two categorical variables in below plot which gives good way to compare the variables.
- In below plot x-axis contains possible gear numbers of vehicles and in y-axis distribution of number of **cylinders(cyl)**.

```
#2) Vertical stacked bar plots
pd.crosstab(dat['gear'],dat['cyl']).plot(kind='bar', stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc279c88>
```

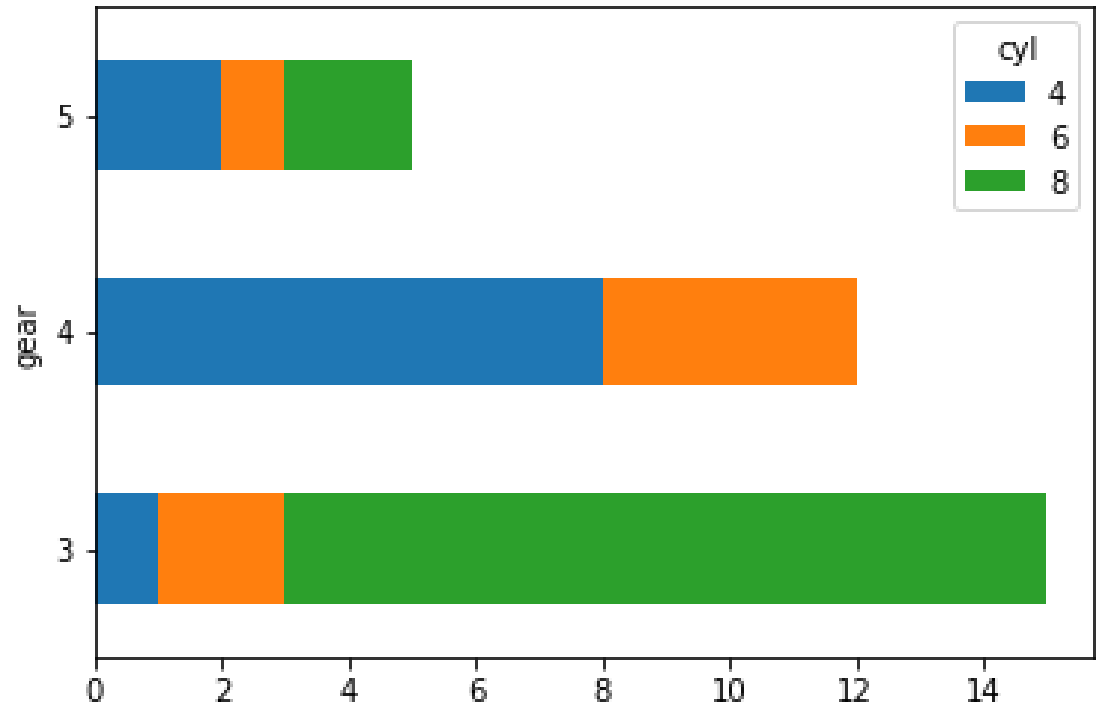


Horizontal Stacked Bar Plot

Horizontal Stacked Bar Plot: We can also plot horizontal bar plot for same example by changing the "**kind**" from to "**bar**" to "**barh**" (h –horizontal).

```
: #3) Horizontal stacked bar plots
  pd.crosstab(dat['gear'],dat['cyl']).plot(kind='barh', stacked=True)
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0xc3705c0>
```

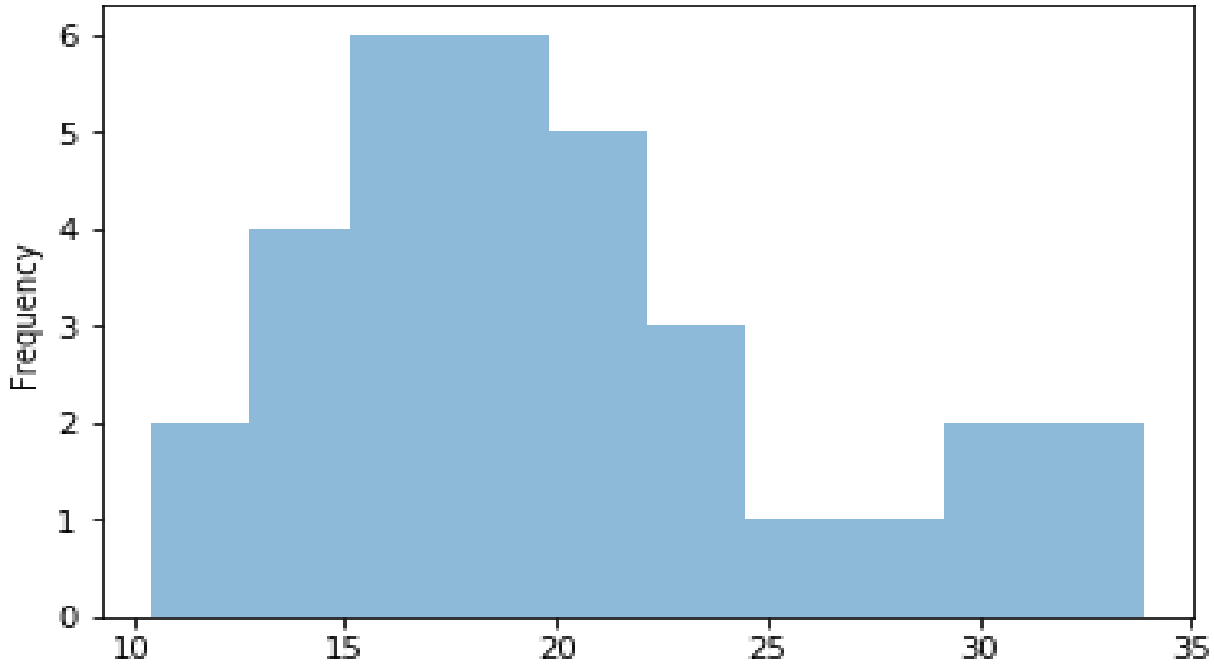


Histogram

- **Histogram:** Histogram is used for density estimation, Histogram is plotted to numeric variable.
- Let us plot histogram of **'mgp'**(mileage) of vehicles in the **mtcars** dataset**.**

```
##########################################
# Histogram - Numeric
##########################################
dat['mpg'].plot(kind='hist', alpha=0.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xcf54f28>
```

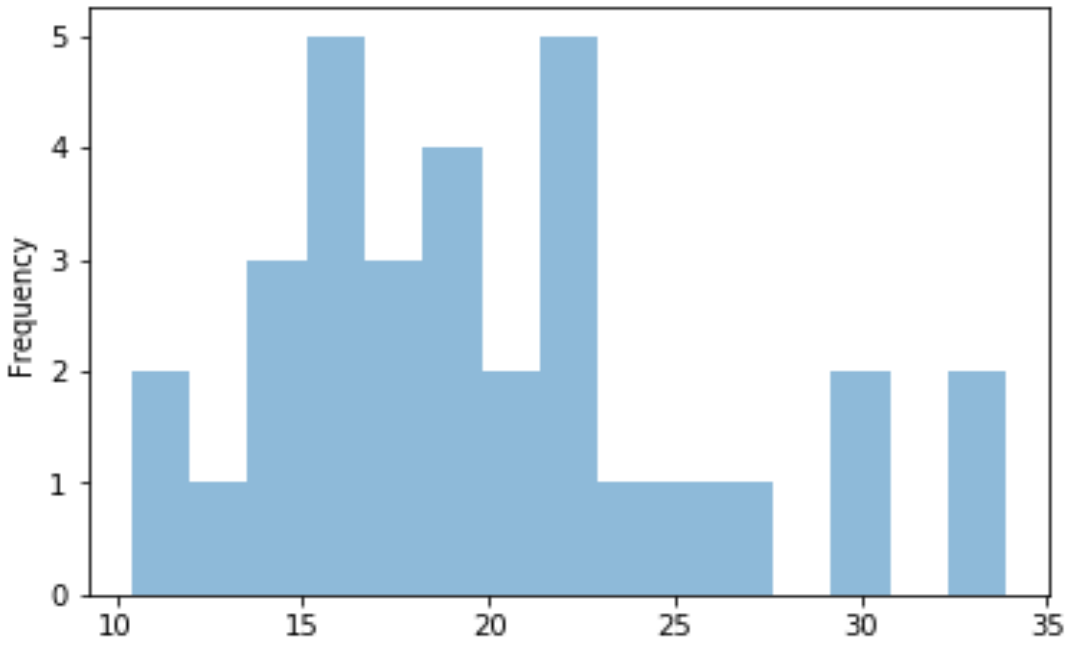


Histogram

- **Histogram using bins:** We can specify the **'bin'** (interval) to the histogram, which tells the number of values which must fall in a particular interval

```
dat['mpg'].plot(kind='hist', alpha=0.5, bins=15)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xdf89e48>
```

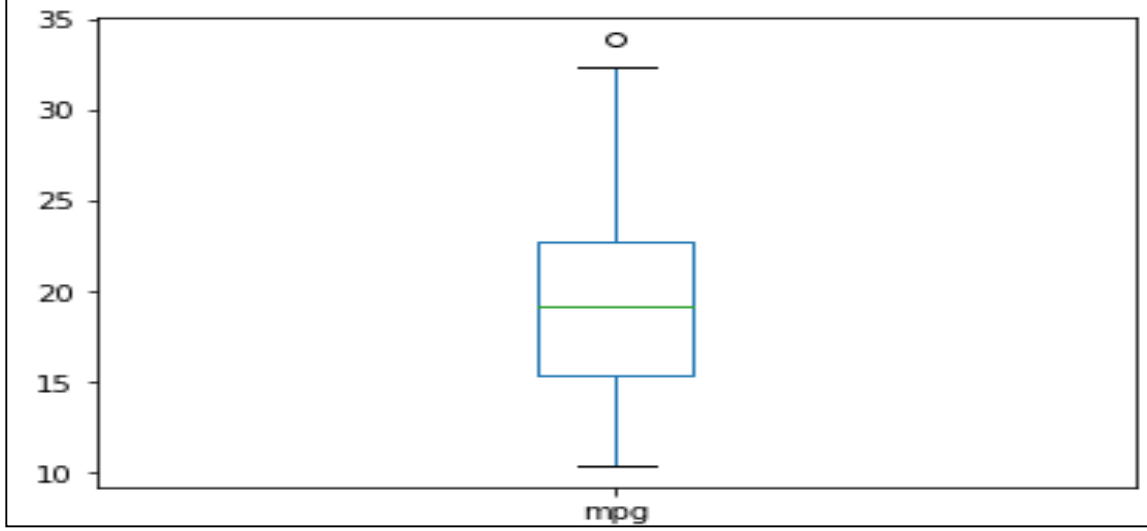


Boxplot

- **Box plot:** We draw boxplot to show the distribution with respect to categories.
- Box represents different quartiles and viscous extends to show the rest of distribution.
- Boxplots are very helpful to “outlier” detection.
- Example: Let us plot a simple box plot of **‘mpg’(mileage)** using **mtcars** dataset.

```
##########################################
#Boxplot - Numeric/Categorical
##########################################
#1) Normal box plot
#dat['mpg'].groupby(dat['gear']).boxplot()
dat['mpg'].plot(kind='box')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xdf930f0>
```



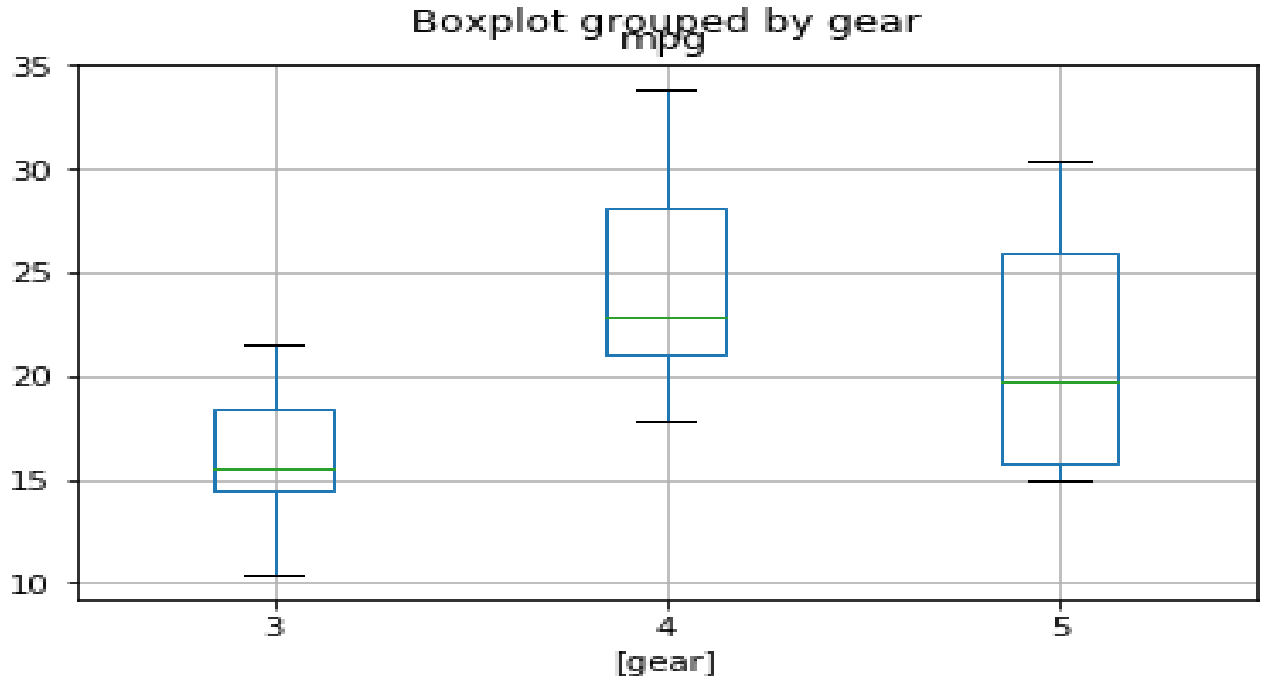
Boxplot

- **Box plot by groups:** We can plot the box plots by the groups they belongs. It gives good visualization for comparing distributions of different groups.

```
#2) Boxplot by groups
dat[['mpg','gear']].boxplot(by='gear')
```

```
C:\Users\hi\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:57: FutureWarning: reshape is deprecated and will raise in a
subsequent release. Please use .values.reshape(...) instead
  return getattr(obj, method)(*args, **kwds)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xabff908>
```

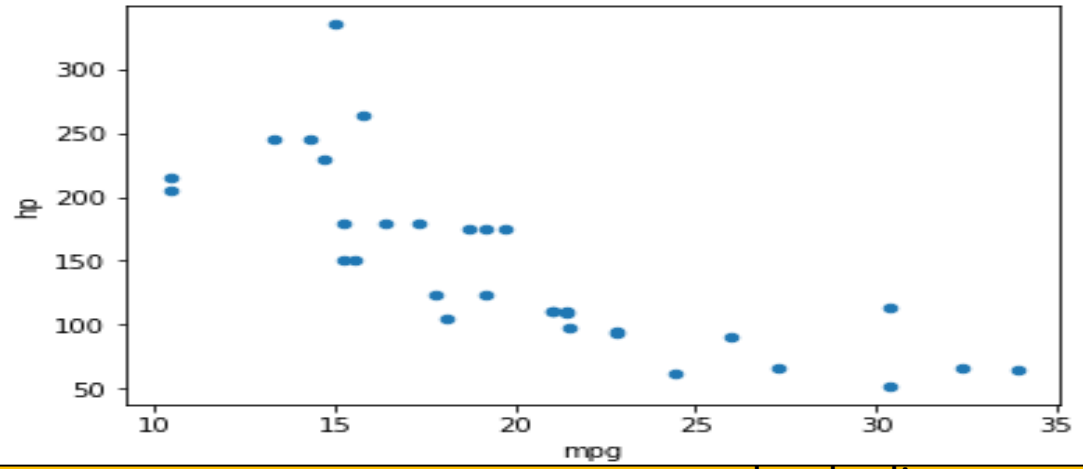


Scatter Plot

- **Scatter plot:** Scatter plot is used to visualize how one variable is varying with other i.e. correlation between variables.
- Scatter plot can be plotted for **Numeric** vs **Numeric** or **Numeric** vs **Numeric** vs **Categorical** and Others.
- **Simple Scatter plot**: Let us plot a simple scatter plot using variables from mtcars. We are plotting **'mpg'**(Mileage) vs **'hp'**(Horse Power).

```
##########################################
#Scatter plot - Numeric/Numeric, Numeric/Numeric/Categorical
##########################################
#1)Normal scatter plot
dat.plot(kind='scatter', x='mpg', y='hp')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xaa16748>
```

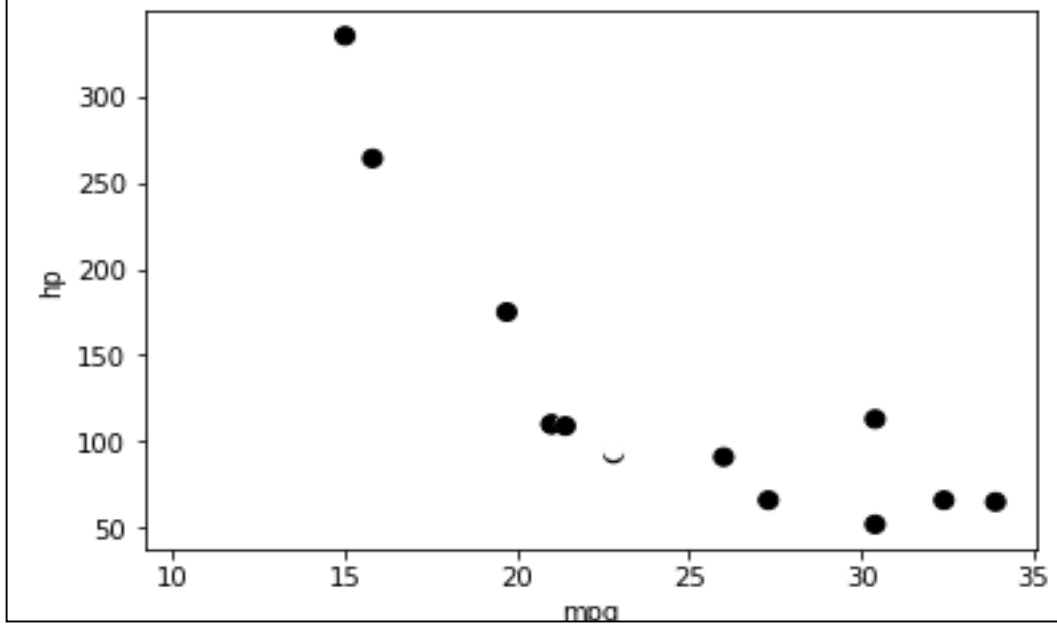


Scatter Plots

- **Scatter plot with 3 variables:** We can plot scatter plot with more than two variables. The following chart contains variables ‘mgp’(Mileage), ‘hp’(Horse power) and ‘am’(Transmission 0 = automatic, 1 = manual).

```
#2) Scatter plot with 3 variables
dat.plot(kind='scatter', x='mpg', y='hp', c=dat['am'].astype(str), s=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xbdfad30>
```

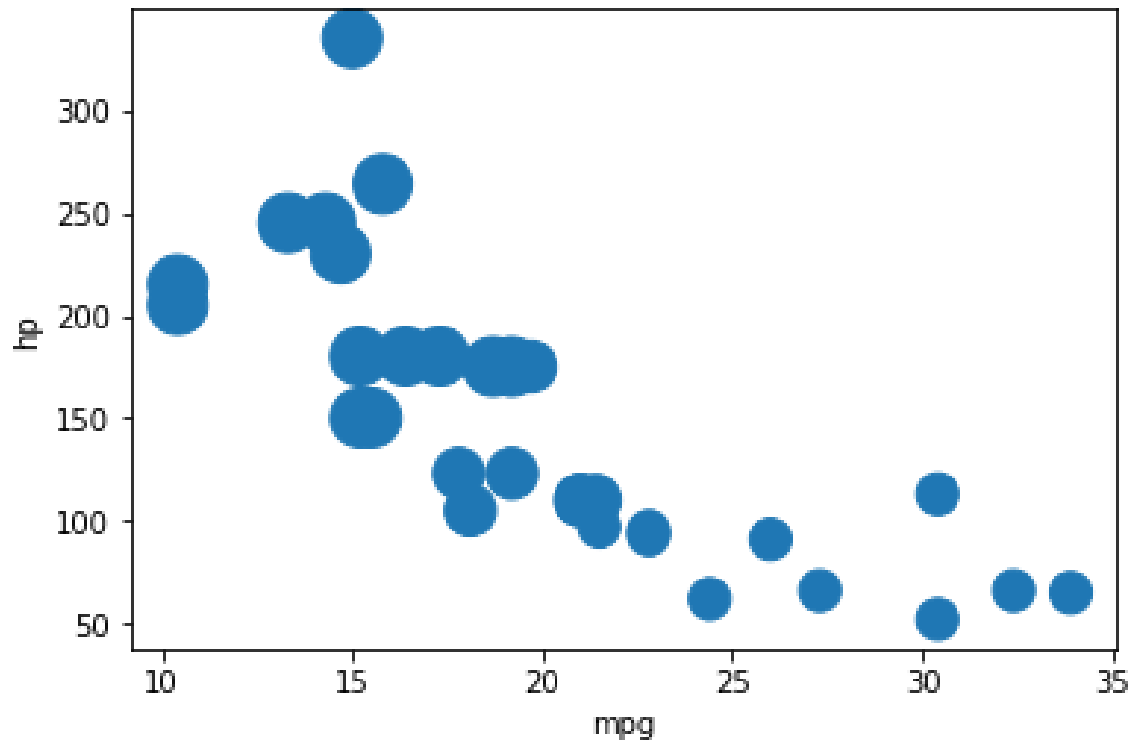


Scatter Plot

- **Scatter Plot Similar to Bubble chart:** We can plot bubble chart using scatter plot as follows.

```
#3)Scatter plot like bubble chart
dat.plot(kind='scatter', x='mpg', y='hp', s=dat['cyl']*50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xcb93da0>
```

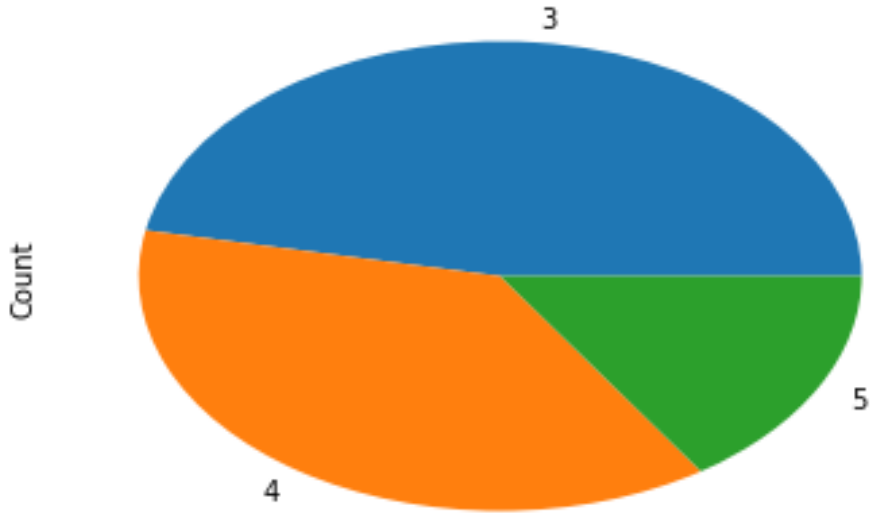


Pie Chart

- **Pie Chart:** It depicts the numerical proportion of data. The proportions are showed using wedges.
- **Simple Pie Chart:** Let us plot a pie chart of gears. The plot shows the proportion of each gear types.

```
In [6]: ########################################
        #Pie chart - Categorical
        ########################################
        #1)Normal pie chart
        d = pd.crosstab(dat['gear'],columns="Count")['Count']
        d.plot(kind='pie')
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x9e7ddd8>
```

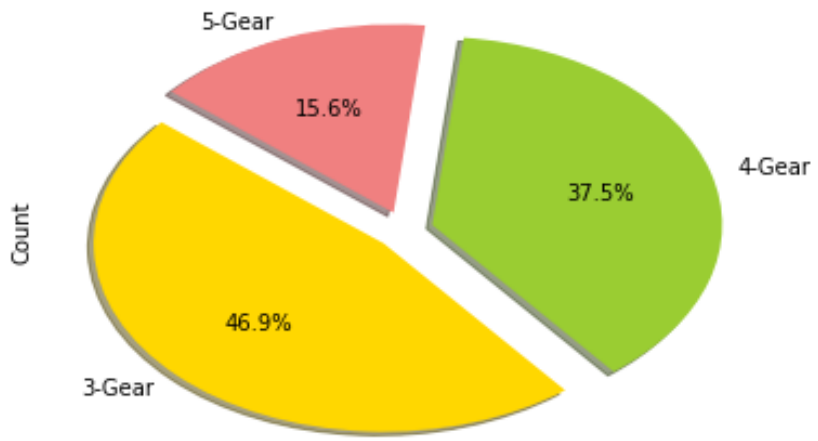


Pie Chart

- **Exploded Pie Chart:** Pie chart created with Separating the wedges of the chart.

  Following chart is an example for exploded pie chart.

```
In [41]: #2) Exploded pie chart
         labels = ['3-Gear', '4-Gear', '5-Gear']
         colors = ['gold', 'yellowgreen', 'lightcoral']
         explode = (0.1, 0.1, 0.1)
         d.plot(kind='pie', explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0xcbfc9b0>
```

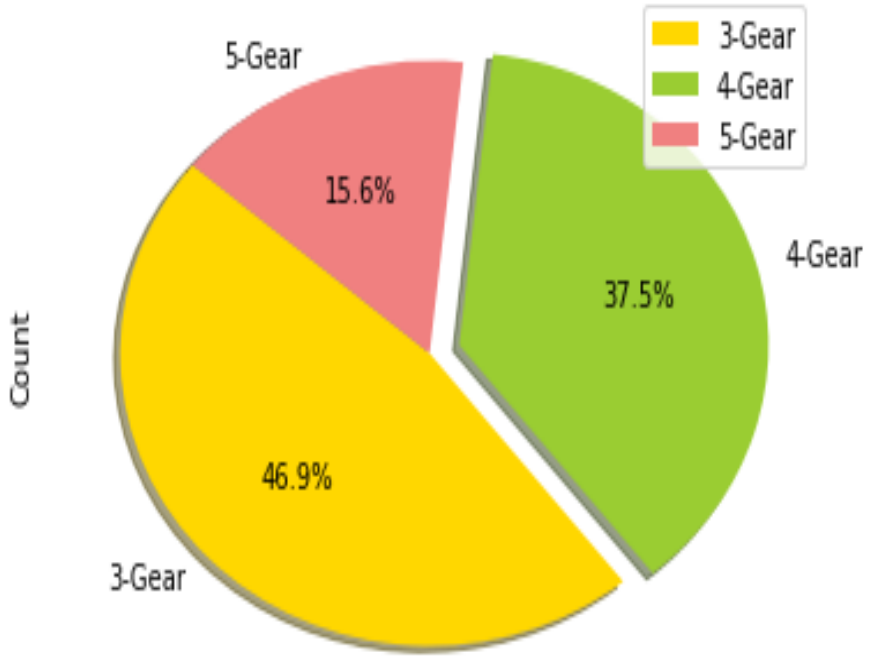


Pie Chart

- **Single Exploded Pie Chart:** Pie chart created with Separating a single wedge of the chart. Following chart is an example for single exploded pie chart.

```
#3) Single exploded pie chart
explode = (0, 0.1, 0)
d.plot(kind='pie', explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140, legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xccdd668>
```

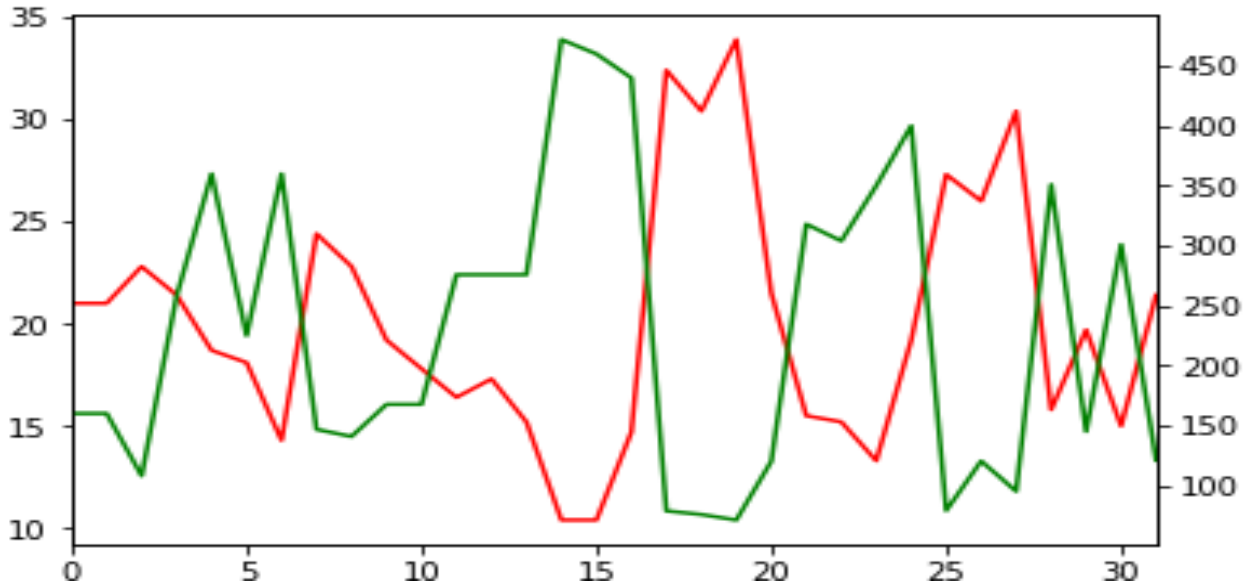


Adding Secondary Axis

- **Secondary Axis:** If we want to compare 2 line charts then We can add second line chart as secondary axis in same chart.
- **Example:** If we need line charts of both ‘mpg’ and ‘disp’ in a same chart then we can make ‘disp’ as secondary axis and get then in same chart.

```
dat['mpg'].plot(kind='line',style='r')
dat['disp'].plot(secondary_y=True, style='g')
```

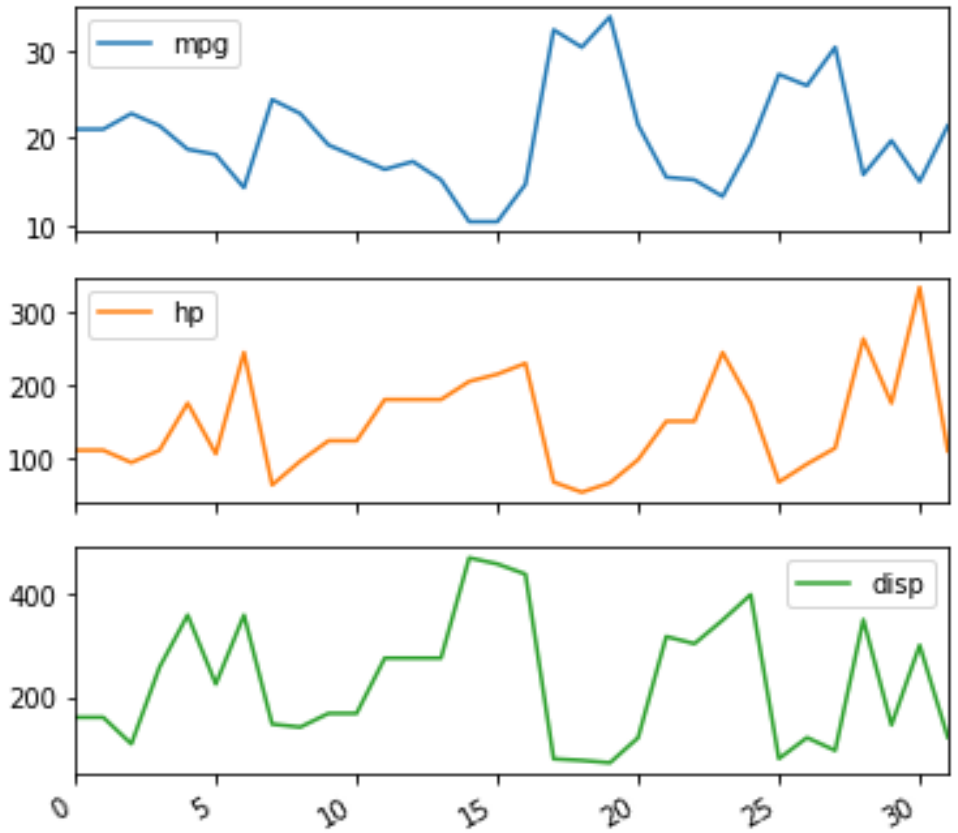
```
<matplotlib.axes._subplots.AxesSubplot at 0xcd77400>
```



Adding Multiple Charts

- **Multiple Charts:** We can plot multiple plots one below the other as subplots.
- **Example:** in below example we plotted line charts **'mpg', 'hp',** and **'disp'** as sub plots in a single chart.

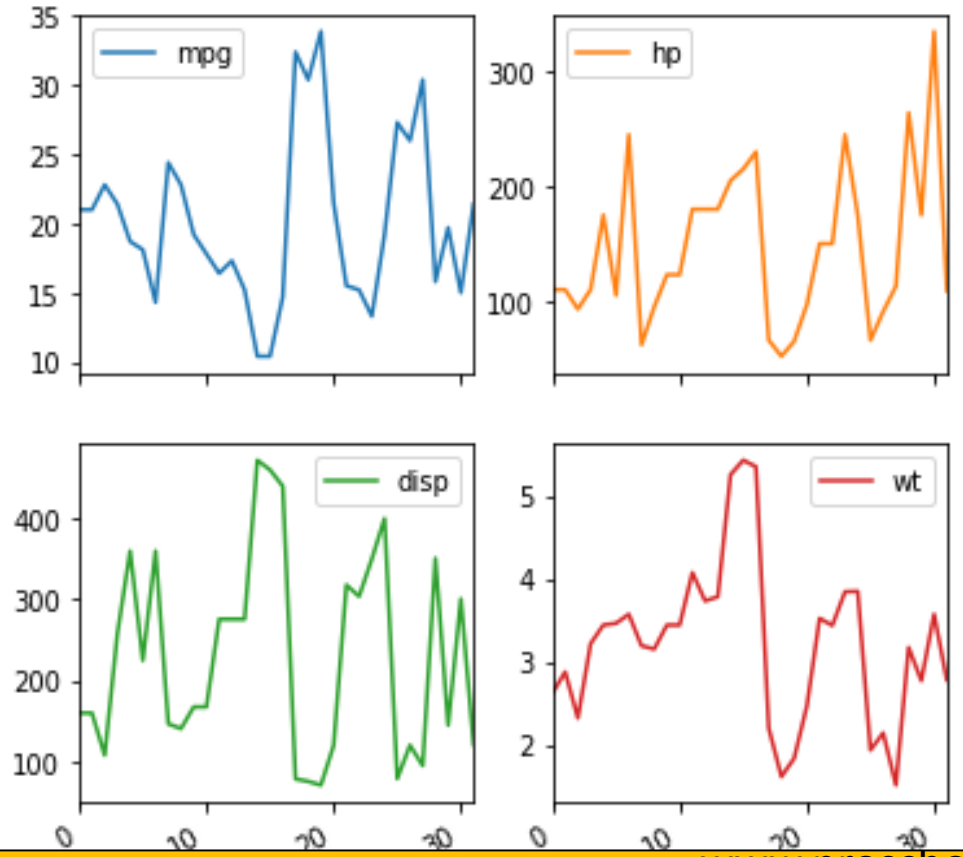
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000000000CD8FBA8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CE75278>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000000000CE2E198>], dtype=object)
```



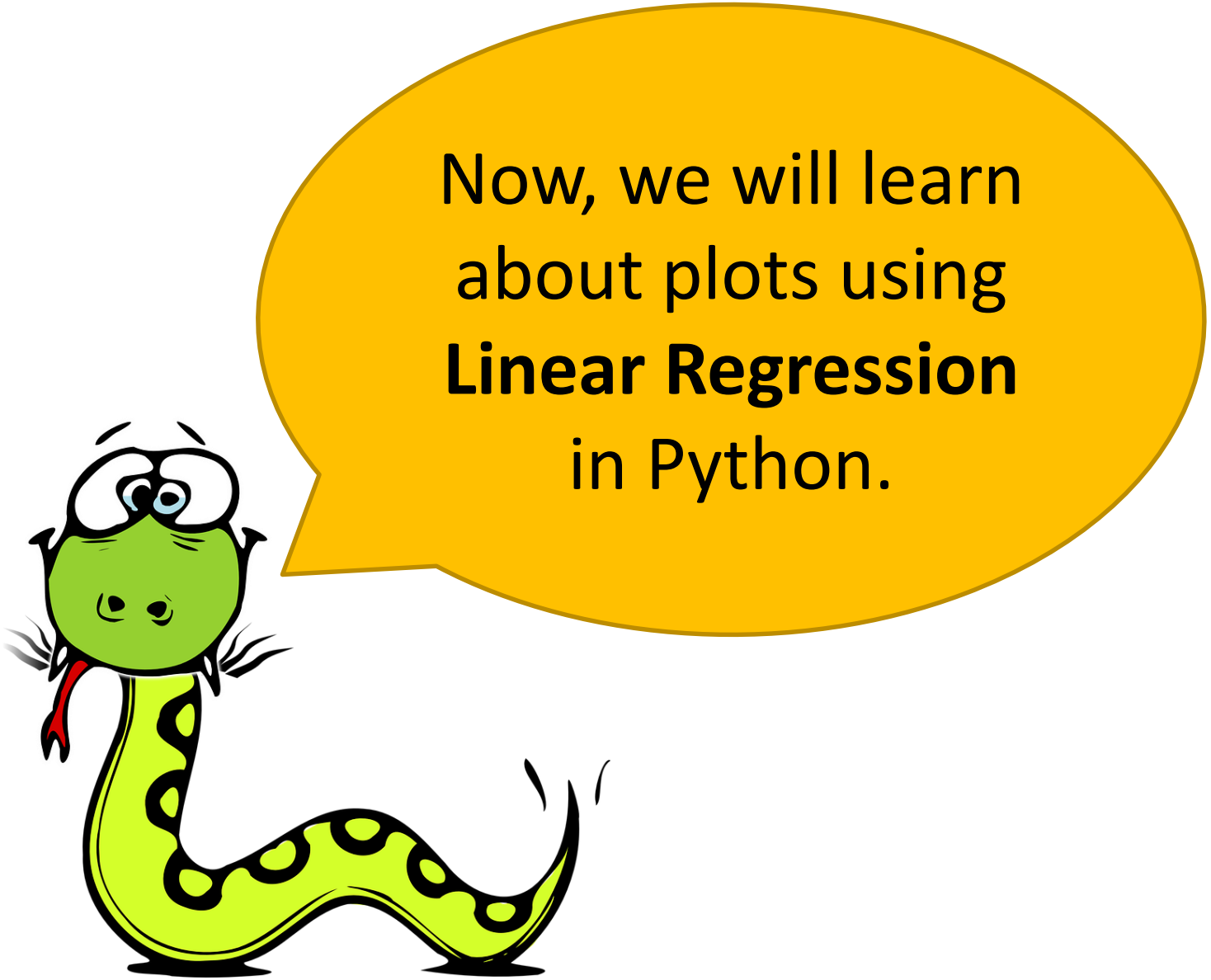
Plot in Layout

- **Plots in Layout:** We can plot multiple plots side by side as subplots.
- **Example:** in below example we plotted line charts of **'mpg', 'hp', 'disp' and 'wt'** as sub plots in a **2X2 layout**

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000CC26908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C1CCC18>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000C34C4A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C2172B0>]], dtype=object)
```



Linear Regression



Load Packages

- Load the following packages,
  - Numpy – For numeric functions in python
  - Pandas – Package to work with data preparations
  - Matplotlib.Pyplot – For plots
  - Sklearn – Scikit learn has several functions for machine learning algorithms
  - Seaborn – For plots
  - Statsmodels.formula.api – Linear regression function

In [23]:
```
#importing packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
import seaborn
import statsmodels.formula.api as sm
```

Load Data

proschool
An IMS Initiative

- Read the data available in csv file. Data can be loaded from any file types like excel, csv, text, sql etc.

In [15]:
```
#Load data
data = pd.read_csv("E:/IMS/IMS Python/Python/Python/Data Files/LR_1.csv", header=0)
print("Data shape : ",data.shape)
print(data.head())
print(data.dtypes)
```

```
Data shape :  (10, 3)
   House Price  Square Feet City
0          245         1400    N
1          312         1600    Y
2          279         1700    N
3          308         1875    Y
4          199         1100    N
House Price     int64
Square Feet     int64
City           object
dtype: object
```

Dummy Creation

- Get all the categorical variables in the data and convert them into dummies.

```
In [4]: #Get all categorical variables and create dummies
        obj = data.dtypes == np.object
        print(obj)
        dummydf = pd.DataFrame()

        for i in data.columns[obj]:
            dummy = pd.get_dummies(data[i], drop_first=True)
            dummydf = pd.concat([dummydf, dummy], axis=1)
```

```
House Price    False
Square Feet    False
City            True
dtype: bool
```

- The “dtypes” gets the data type, using which we pass this column to the “get_dummies” function to convert the categories to dummy columns.
- “drop_first” drops the first category in order to avoid multicollinearity problem
- “pd.concat” combines all the dummy columns for all the categorical variables

Merge Data

- Merge the dummy data with the original data

```
In [5]: #Merge the dummy and dataset
        data1 = data
        data1 = pd.concat([data1,dummydf], axis=1)
        print("head \n",data1.head())
        obj1 = data1.dtypes == np.object
        data1 = data1.drop(data1.columns[obj1], axis=1)
        print("head after removal \n",data1.head())
```

```
head
    House Price  Square Feet City  Y
0           245          1400    N  0
1           312          1600    Y  1
2           279          1700    N  0
3           308          1875    Y  1
4           199          1100    N  0
head after removal
    House Price  Square Feet  Y
0           245          1400  0
1           312          1600  1
2           279          1700  0
3           308          1875  1
4           199          1100  0
```

- drop" function is used to drop the original column of the categorical variable
- To view final data,

Dependent & Independent Variables

- Separate the dependent & independent variables into two objects.

```
In [4]: #Get all categorical variables and create dummies
        obj = data.dtypes == np.object
        print(obj)
        dummydf = pd.DataFrame()

        for i in data.columns[obj]:
            dummy = pd.get_dummies(data[i], drop_first=True)
            dummydf = pd.concat([dummydf, dummy], axis=1)
```

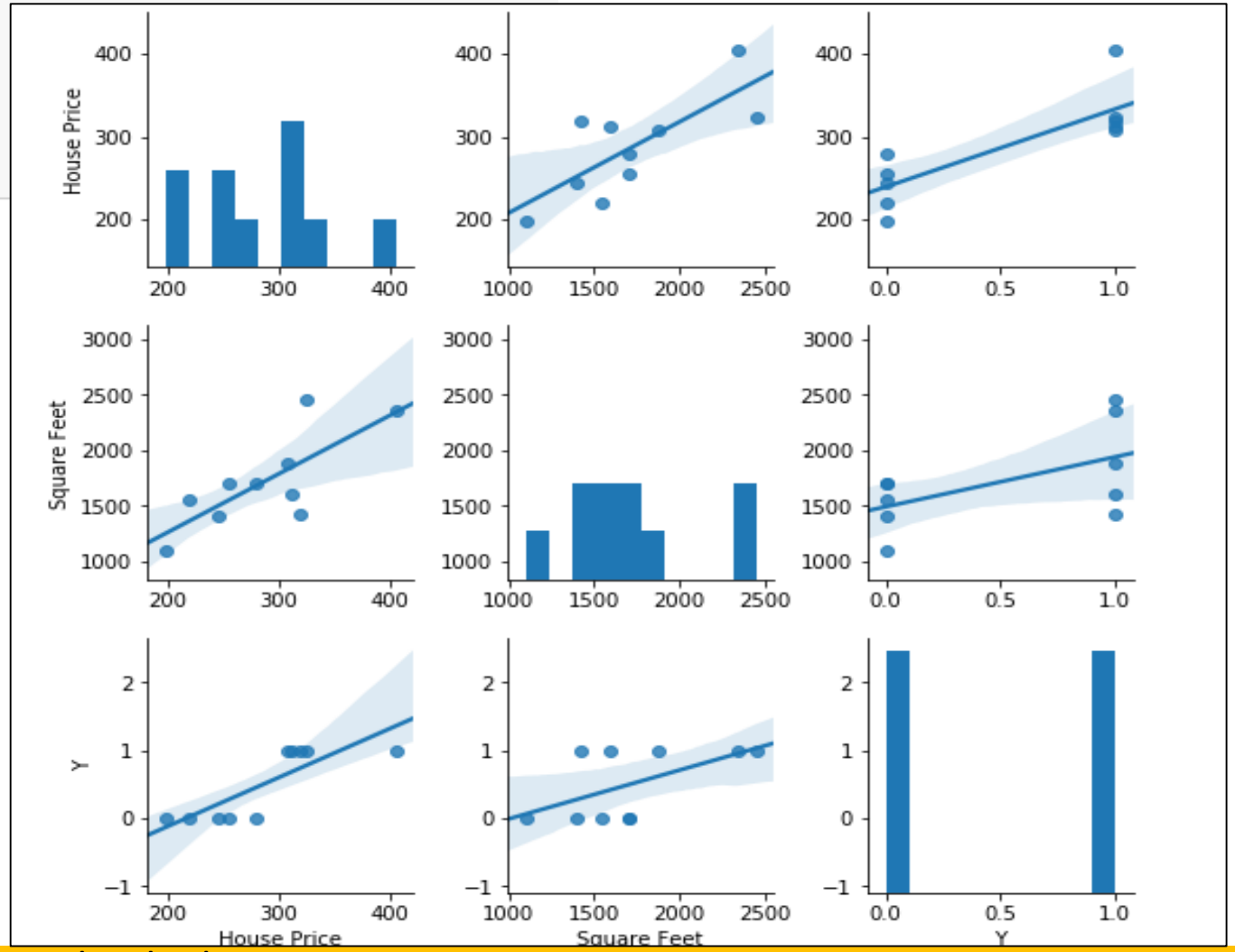
```
House Price    False
Square Feet    False
City            True
dtype: bool
```

Scatter Plots

- Scatter plots can be used to check the relation of the variables

```
#Declare the dependent variable and create your independent and dependent datasets
dep = 'House Price'
X = data1.drop(dep, axis=1)
Y = data1[dep]

#Scatter plots
seaborn.pairplot(data1, kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0xdc99ac8>
```



Train & Test

- Split the data into train & test for model building and validating the model
- We us the model_selection in sklearn package to split the data into train & test
- The “test_size” is used to specify size of the test data, in our case 20%
- “random_state” is used to generate the same set of sample which helps in replicating the results (It acts like seed to generate random number.

```
In [6]: import sklearn.cross_validation
        #Split into train and test
        X_train, X_test, Y_train, Y_test = sklearn.cross_validation.train_test_split(X, Y, test_size=0.20, random_state=5)
```

Model Building

- Use the OLS function in the statsmodels. formual.api to build a linear regression model

In [7]:
```
#Run model
lm = sm.OLS( Y_train, X_train ).fit()
lm.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	House Price	**R-squared:**	0.979
Model:	OLS	**Adj. R-squared:**	0.973
Method:	Least Squares	**F-statistic:**	143.1
Date:	Tue, 19 Dec 2017	**Prob (F-statistic):**	8.65e-06
Time:	23:00:09	**Log-Likelihood:**	-41.550
No. Observations:	8	**AIC:**	87.10
Df Residuals:	6	**BIC:**	87.26
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Square Feet	0.1567	0.019	8.163	0.000	0.110	0.204
Y	29.5832	43.518	0.680	0.522	-76.900	136.067

Omnibus:	6.866	**Durbin-Watson:**	2.239
Prob(Omnibus):	0.032	**Jarque-Bera (JB):**	1.994
Skew:	-1.151	**Prob(JB):**	0.369
Kurtosis:	3.826	**Cond. No.**	4.38e+03

Error Calculation

- Error can calculated by predicting the values from the model and subtracting from the actual values
- The “lm.predict” function can be used for predictions

```
In [20]: pred_train = lm.predict(X_train)
         err_train = pred_train - Y_train

         #Predict
         pred_test = lm.predict(X_test)
         err_test = pred_test - Y_test
```

Model Performance

- We can check the root mean square error to see how the model is performing
- This can be calculated on both the train and test and compared to see how the model is performing on the test dataset

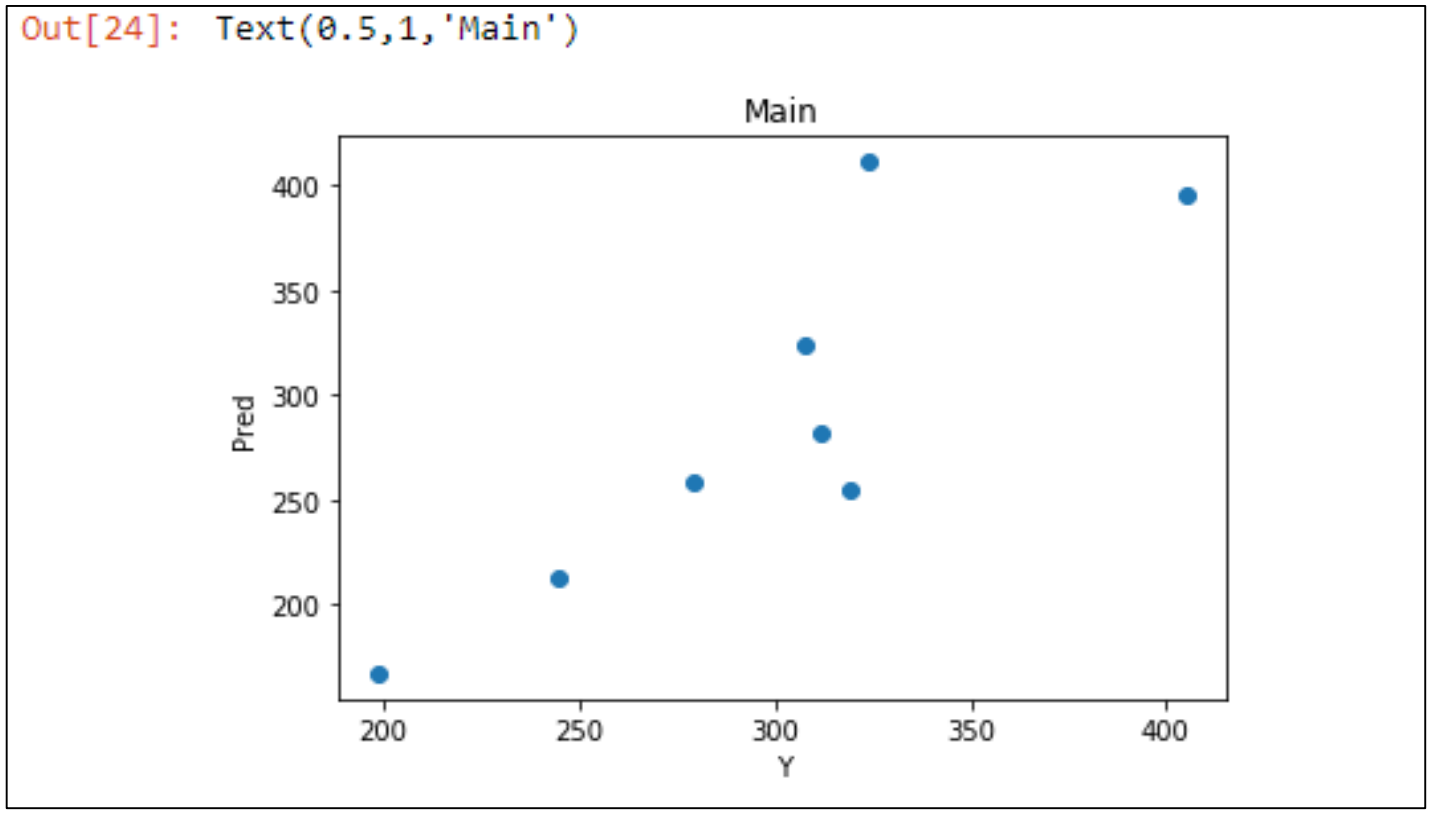
```
In [24]: #Actual vs predicted plot
         plt.scatter(Y_train, pred_train)
         plt.xlabel('Y')
         plt.ylabel('Pred')
         plt.title('Main')
```

Model Plots

- Plot the model for actual and predicted

```
In [24]: #Actual vs predicted plot
         plt.scatter(Y_train, pred_train)
         plt.xlabel('Y')
         plt.ylabel('Pred')
         plt.title('Main')
```

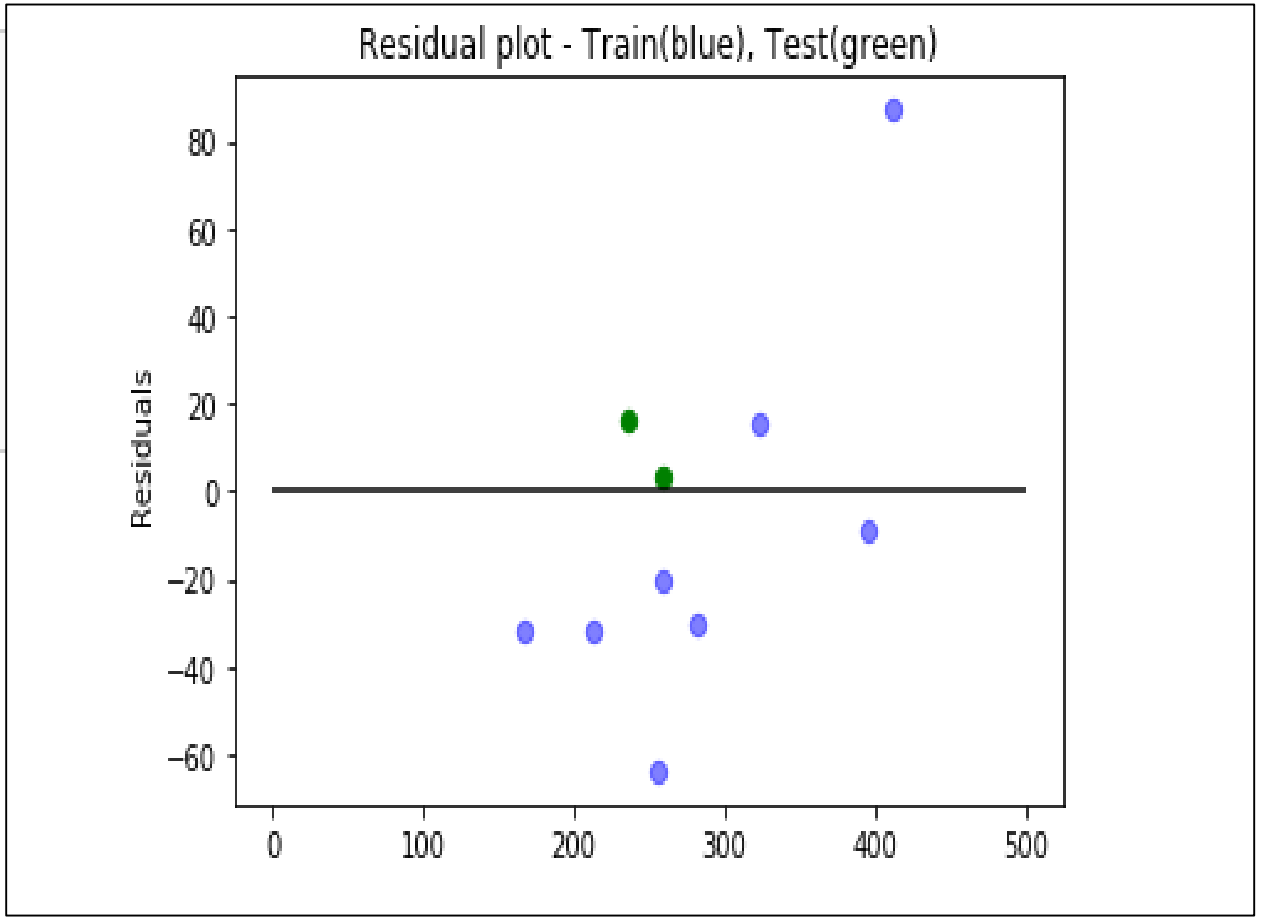
```
Out[24]: Text(0.5,1,'Main')
```



Residual Plots

Plot the residuals and the predicted values

```
In [26]: #Residual plots
         plt.scatter(pred_train, err_train, c="b", s=40, alpha=0.5)
         plt.scatter(pred_test,err_test, c="g", s=40)
         plt.hlines(y=0, xmin=0, xmax=500)
         plt.title('Residual plot - Train(blue), Test(green)')
         plt.ylabel('Residuals')

Out[26]: Text(0,0.5,'Residuals')
```



Logistic Regression

Load Packages

- Load the following packages,
  - Numpy – For numeric functions in python
  - Pandas – Package to work with data preparations
  - Sklearn – Scikit learn has several functions for machine learning algorithms
  - Statsmodels.formula.api – Logistic regression function
  - Cross_validation – Module to create train & test split

```
In [1]: #importing packages
        import pandas as pd
        import numpy as py
        from sklearn.linear_model import LogisticRegression
        from sklearn.cross_validation import train_test_split
        import statsmodels.api as sm
```

User Defined Function

Define a function which can calculate the measures based on the confusion matrix

```
def measures(Technique, Actual, Predicted, cutoff=0.5):
    import pandas as pd
    import numpy as np
    from sklearn import metrics

    confusionmat = metrics.confusion_matrix(Actual, Predicted)

    TP = confusionmat[0,0]
    TN = confusionmat[1,1]
    FP = confusionmat[0,1]
    FN = confusionmat[1,0]
    P = TP+FN
    N = TN+FP

    #Measures
    Acc=np.round((TP+TN)/(P+N),4)
    KS=np.round(abs((TP/P)-(FP/N)),4)
    Precision = np.round((TP/(TP+FP)),4)
    Recall = np.round((TP/(TP+FN)),4)

    Results = pd.DataFrame(columns = ['Technique', 'Cutoff', 'Accuracy', 'KS', 'Precision', 'Recall'])
    Results = Results.append(pd.DataFrame([Technique , cutoff, Acc, KS, Precision, Recall],
                                          index=['Technique' ,'Cutoff', 'Accuracy', 'KS', 'Precision', 'Recall']).T)
    return(Results)
```

User Defined Function

Define a function which can be used to find the best threshold

```
#########################################################################################################
# Function Name    : bestcutoff
# Function Desc    : Iterates over different threshold values to find the optimal cutoff
# Parameters       : Technique - a string giving the technique name
#                    actual    - The actual target variable
#                    pred      - The predicted value for teh target variable
#                    cuts      - The jumps required for thresholds to be generated. Value between 0 to 100
#########################################################################################################
def bestcutoff(actual, pred, Technique, cuts=4):
    import numpy as np
    seq = np.round(np.append(np.arange(0,100,cuts)/100,1),2)
    cutdata = pd.DataFrame(columns = ['Technique', 'Cutoff', 'Accuracy', 'KS', 'Precision', 'Recall'])
    final = pd.DataFrame(columns = ['Technique', 'Accuracy', 'KS', 'Precision', 'Recall'])
    for i in seq:
        predicted= pred.loc[:,0].map(lambda x: 1 if x > i else 0)
        temp1 = measures(Technique, actual, predicted, i)
        cutdata = cutdata.append(temp1)

    cutdata = cutdata.set_index(np.arange(len(seq)))

    temp=[]
    for j in range(2,6):
        temp = np.append(temp,cutdata.iloc[cutdata.iloc[:,j].idxmax(),1])

    listval = list(np.append(Technique, temp))
    final.loc[len(final)]=listval

    print(Technique, "bestcutoff Completed \n")
    print("################################### \n")

    return(final)
```

Load data

- Read the data available in csv file. Data can be loaded from any file types like excel, csv, text, sql etc.
- To see the dimension i.e. the rows & columns of the data use the shape attribute
- To see the top 5 rows of your data use head function,

```
#Load data
data = pd.read_csv("E:/IMS/IMS Python/Python/Python/Data Files/binary.csv", header=0)
print("Shape of Data : ",data.shape)
print("Sample Data \n",data.head())
```

```
Shape of Data :  (400, 4)
Sample Data
    admit  gre   gpa  rank
0      0  380  3.61     3
1      1  660  3.67     3
2      1  800  4.00     1
3      1  640  3.19     4
4      0  520  2.93     4
```

Data Type Coversion

- Convert categorical variables with numeric values to str type. Get all categorical variables and create dummies

```
#Convert categorical variables with numeric values to str type
data['rank'] = data['rank'].astype(str)

#Get all categorical variables and create dummies
obj = data.dtypes == py.object
dep = 'admit'
obj[dep]=False
dummydf = pd.DataFrame()

for i in data.columns[obj]:
    dummy = pd.get_dummies(data[i], drop_first=True)
    dummydf = pd.concat([dummydf, dummy], axis=1)
```

- The “dtypes” gets the data type, using which we pass this column to the “get_dummies” function to convert the categories to dummy columns.
- “drop_first” drops the first category in order to avoid multicollinearity problem
- “pd.concat” combines all the dummy columns for all the categorical variables

Merge Data

- Merge the dummy data with the original data

```
#Merge the dummy and dataset
data1 = data
data1 = pd.concat([data1,dummydf], axis=1)
```

- "drop" function is used to drop the original column of the categorical variable
- Declare the dependent variable and create your independent and dependent datasets

```
#Declare the dependent variable and create your independent and dependent datasets
dep='admit'
obj1 = data1.dtypes == py.object
X = data1.drop(data1.columns[obj1], axis=1)
X = X.drop([dep], axis=1)
```

Column Names

- Appending "V" to columns in order to avoid numeric column names generated while creating dummies

```
#Appending "V" to columns in order to avoid numeric column names generated while creating dummies
X.columns = 'V_'+X.columns
Y = data1[dep]
X.columns
```

```
Out[11]: Index(['V_gre', 'V_gpa', 'V_2', 'V_3', 'V_4'], dtype='object')
```

Train & Test

- Split the data into train & test for model building and validating the model
- We us the model_selection in sklearn package to split the data into train & test
- The “test_size” is used to specify size of the test data, in our case 20%
- “random_state” is used to generate the same set of sample which helps in replicating the results

In [12]:
```
#Split into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)

print('Train Data Size - ', X_train.shape[0], '\n')
print('Test Data Size - ', X_test.shape[0], '\n')
```

```
Train Data Size -  320

Test Data Size -  80
```

Model Building

Use the Logit function in the statsmodels.formual.api to build a logistic regression model

In [13]:
```
#Run Model
modLR = sm.Logit(pd.DataFrame(Y_train), X_train, family=sm.families.Binomial())
result = modLR.fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.580186
         Iterations 6
```

Logit Regression Results

Dep. Variable:	admit	**No. Observations:**	320
Model:	Logit	**Df Residuals:**	315
Method:	MLE	**Df Model:**	4
Date:	Tue, 12 Dec 2017	**Pseudo R-squ.:**	0.06951
Time:	23:01:58	**Log-Likelihood:**	-185.66
converged:	True	**LL-Null:**	-199.53
		LLR p-value:	1.409e-05

	coef	std err	z	P>|z|	[0.025	0.975]
V_gre	0.0016	0.001	1.399	0.162	-0.001	0.004
V_gpa	-0.1851	0.215	-0.861	0.389	-0.607	0.236
V_2	-0.9064	0.328	-2.767	0.006	-1.549	-0.264
V_3	-1.4479	0.359	-4.035	0.000	-2.151	-0.745
V_4	-2.3167	0.510	-4.540	0.000	-3.317	-1.316

Odds Ratio

To get the odds ratio from the estimates obtained take the exponential of the estimates

```
In [14]: #odds ratio
         py.exp(result.params)

Out[14]: V_gre     1.001635
         V_gpa     0.831012
         V_2       0.403956
         V_3       0.235069
         V_4       0.098600
         dtype: float64
```

Model Performance

- For cutoff=0.5 lets check the model performance
- “predict” function is used to get the probabilities

In [15]:
```
#Predict and get the measures for train and test
cutoff = 0.5

pred_train = pd.DataFrame(result.predict(X_train))
predLR = pred_train.loc[:,0].map(lambda x: 1 if x > cutoff else 0)
ResLR = measures('Logistic', Y_train, predLR, cutoff)
ResLR
```

Out[15]:

	Technique	Cutoff	Accuracy	KS	Precision	Recall
0	Logistic	0.5	0.6938	0.2543	0.8904	0.7249

- Apply the model with the above cut off on the test dataset

In [16]:
```
pred_test = pd.DataFrame(result.predict(X_test))
tpredLR = pred_test.loc[:,0].map(lambda x: 1 if x > cutoff else 0)
tResLR = measures('Logistic', Y_test, tpredLR, cutoff)
tResLR
```

Out[16]:

	Technique	Cutoff	Accuracy	KS	Precision	Recall
0	Logistic	0.5	0.7	0.3143	0.9259	0.7143

Optimal Cut-off

- Optimal cutoff - Helps you find the optimal cutoff value by each measure

In [17]:
```
#Optimal cutoff - Helps you find the optimal cutoff value by each measure
optcut = pd.DataFrame(columns = ['Technique', 'Accuracy', 'KS', 'Precision', 'Recall'])
optcut = optcut.append(bestcutoff(Y_train, pred_train, 'Logistic',cuts=4))
optcut
```

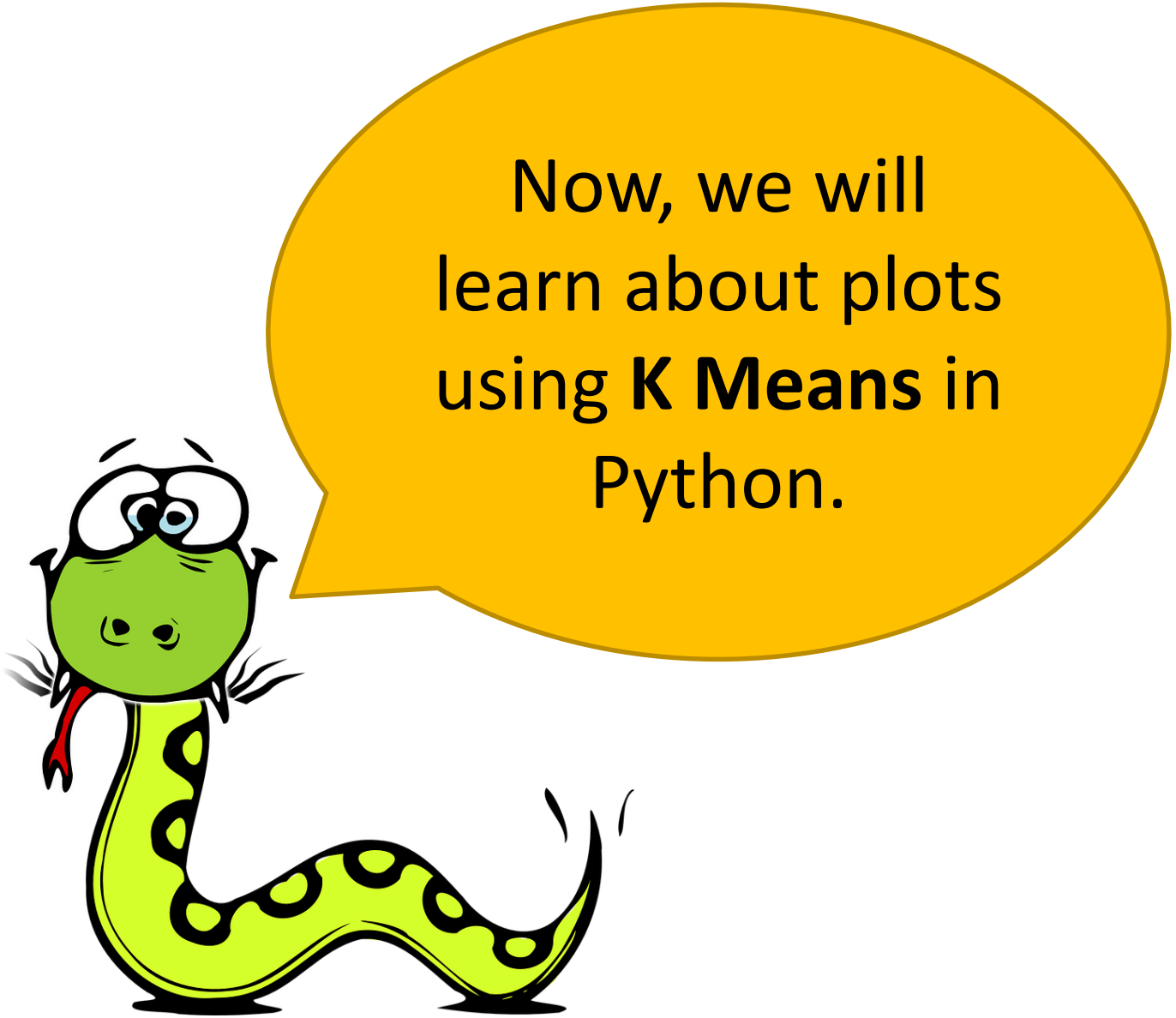
Out[17]:

	Technique	Accuracy	KS	Precision	Recall
0	Logistic	0.44	0.64	0.68	0.12

- The above values indicate at which particular cut-off the measures high i.e. for example at the cutoff=0.44 the accuracy measure is high for this model, Similarly at cutoff=0.68 precision is high

- Based on the business problem the appropriate measure needs to be selected

K Means



Load Package

- Load the following packages,
  - Numpy – For numeric functions in python
  - Pandas – Package to work with data preparations
  - Sklearn – Scikit learn has several functions for machine learning algorithms
  - Sklearn.cluster – Kmeans module
  - preprocessing – Module to scale your data
  - Linkage, dendrogram = Modules to use the linkage & dendrogram
  - pdist – Used to calculate the pairwise distances
  - cdist - Computes distance between each pair of the two collections of inputs.

```
In [12]: #importing packages
         import pandas as pd
         import numpy as np
         import os as os
         from sklearn.cluster import KMeans
         from sklearn import preprocessing
         import matplotlib.pyplot as plt
         import matplotlib
         %matplotlib inline
         from scipy.cluster.hierarchy import linkage, dendrogram
         from scipy.spatial.distance import pdist
         from scipy.spatial.distance import cdist
```

Load Data

- Read the data available in csv file. Data can be loaded from any file types like excel, csv, text, sql etc.
- To see the dimension i.e. the rows & columns of the data use the shape attribute

In [3]:
```
#Reading dataset and getting variables to be considered for clustering
dat = pd.read_csv('E:/IMS/IMS Python/Python/Python/Data Files/europe.csv')
dat1 = dat.drop('Country', axis=1)
dat1.head()
```

Drop the 'Country' column as it may not be useful for clustering. We could analyse the countries after creating the clusters. To see the top 5 rows of your data use head function,

Out[3]:

	Area	GDP	Inflation	Life.expect	Military	Pop.growth	Unemployment
0	83871	41600	3.5	79.91	0.80	0.03	4.2
1	30528	37800	3.5	79.65	1.30	0.06	7.2
2	110879	13800	4.2	73.84	2.60	-0.80	9.6
3	56594	18000	2.3	75.99	2.39	-0.09	17.7
4	78867	27100	1.9	77.38	1.15	-0.13	8.5

Standardizing Variables

proschool
An ims Initiative

Use the preprocessing module for scaling the variables. Scaling means subtracting the mean and dividing the standard deviation for each variable

```
In [6]: #scaling the variables
        X_scaled = preprocessing.scale(dat1)
        X_scaled
```

```
Out[6]: array([[-0.50783522,  0.68390042,  0.11444681,  0.57077826, -1.0243472 ,
                -0.17678894, -1.24552737],
               [-0.83598724,  0.41706139,  0.11444681,  0.48775597, -0.38895239,
                -0.11592718, -0.59244186],
               [-0.34168914, -1.26823778,  0.62425535, -1.36747296,  1.26307411,
                -1.86063119, -0.06997345],
               [-0.67563611, -0.97331043, -0.75951067, -0.68094246,  0.99620829,
                -0.42023602,  1.69335744],
               [-0.5386185 , -0.33430116, -1.05082984, -0.23709251, -0.57957083,
                -0.50138504, -0.30943814],
               [-0.75868454,  0.36088475, -0.39536172,  0.20995061, -0.38895239,
                 0.24924343, -0.83190655],
               [-0.74555673, -0.80478051,  1.20689367, -1.45049525,  0.50060034,
                -1.55632235,  0.56134255],
               [ 1.05639118,  0.29066395, -0.03121277,  0.41112001,  0.50060034,
                -0.09563992, -0.46182476],
               [ 1.1725175 ,  0.43812763, -0.61385109,  0.66018689, -0.13479447,
                -0.64339583, -0.85367607],
```

Determine the Optimal Cluster

- To determine the optimal cluster we need to run Kmeans for several cluster sizes
- KM contains the iterations & 'centroids' contains the centers of each cluster for each variable

```
In [8]: K = range(1,15)
        KM = [KMeans(n_clusters=k).fit(X_scaled) for k in K]
        centroids = [k.cluster_centers_ for k in KM]
        print("KMeans \n",KM)
        print("Centroid \n",centroids)
```

```
KMeans
 [KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=1, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0), KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0), KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0), KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0), KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,

Centroid
 [array([[  8.12841857e-17,  -6.14587746e-17,  -9.11968913e-17,
         -5.51146430e-15,  -7.93016446e-18,   7.93016446e-18,
         -2.69625592e-16]]), array([[-0.18038038, -0.8510092 ,  0.52714896, -0.88636942,  0.43917884,
        -0.77188179,  0.63934999],
       [ 0.13528529,  0.6382569 , -0.39536172,  0.66477706, -0.32938413,
         0.57891134, -0.47951249]]), array([[-0.4493563 , -0.73455971,  0.15693086, -0.65646153,  0.41270406,
        -0.68735155,  0.65023474],
```

Sum of Square

- Calculate the distance of the points from the center of the cluster to calculate the within sum of squares
- Since there is no direct way to get the sum of squares we have to use the below code to obtain them

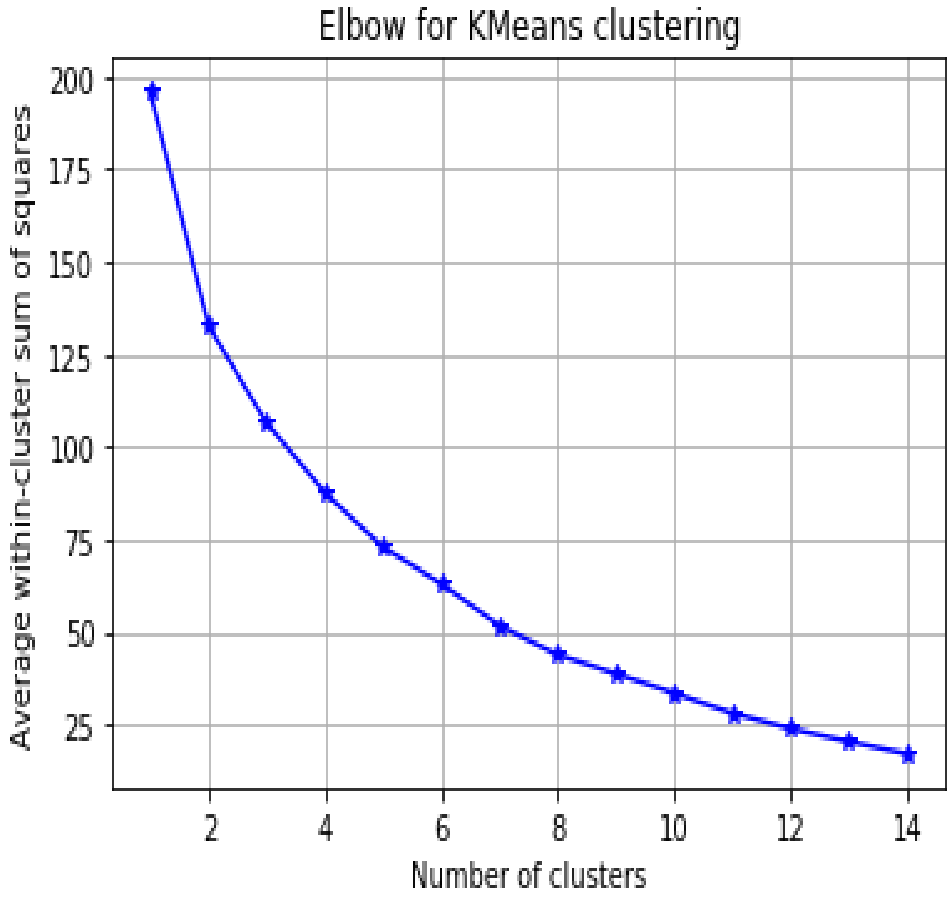
In [9]:
```
#Calculate the distance of the points from the centre of the cluster to calculate the within sum of squares
D_k = [cdist(X_scaled, cent, 'euclidean') for cent in centroids]
cIdx = [np.argmin(D,axis=1) for D in D_k]
dist = [np.min(D,axis=1) for D in D_k]
avgWithinSS = [sum(d)/X_scaled.shape[0] for d in dist]
```

In [10]:
```
# Total with-in sum of square
wcss = [sum(d**2) for d in dist]
tss = sum(pdist(X_scaled)**2)/X_scaled.shape[0]
bss = tss-wcss

kIdx = 10-1
```

Elbow Chart

To determine the optimal cluster size we plot the elbow curve. The within sum of squares is plotted for each of the cluster size and the point at which the curve starts becoming flat that point is considered as optimal point

```
In [13]: # elbow curve
         fig = plt.figure()
         ax = fig.add_subplot(111)
         ax.plot(K, wcss, 'b*-')
         ax.plot(K[kIdx], wcss[kIdx])
         plt.grid(True)
         plt.xlabel('Number of clusters')
         plt.ylabel('Average within-cluster sum of squares')
         plt.title('Elbow for KMeans clustering')

Out[13]: Text(0.5,1,'Elbow for KMeans clustering')
```



Final Cluster

- Based on the elbow curve we could say that cluster=4 we see that change after this point is minimal, so lets fix this as our cluster size
- Run Kmeans after fixing the cluster size and get the cluster names to which the records belong to

In [14]:
```
#Finalize the cluster
kmeans = KMeans(n_clusters=4, random_state=10).fit(X_scaled)
clusters=pd.DataFrame(kmeans.labels_, columns=['Cluster'])
dat = pd.concat([dat, clusters], axis=1)
dat.head()
```

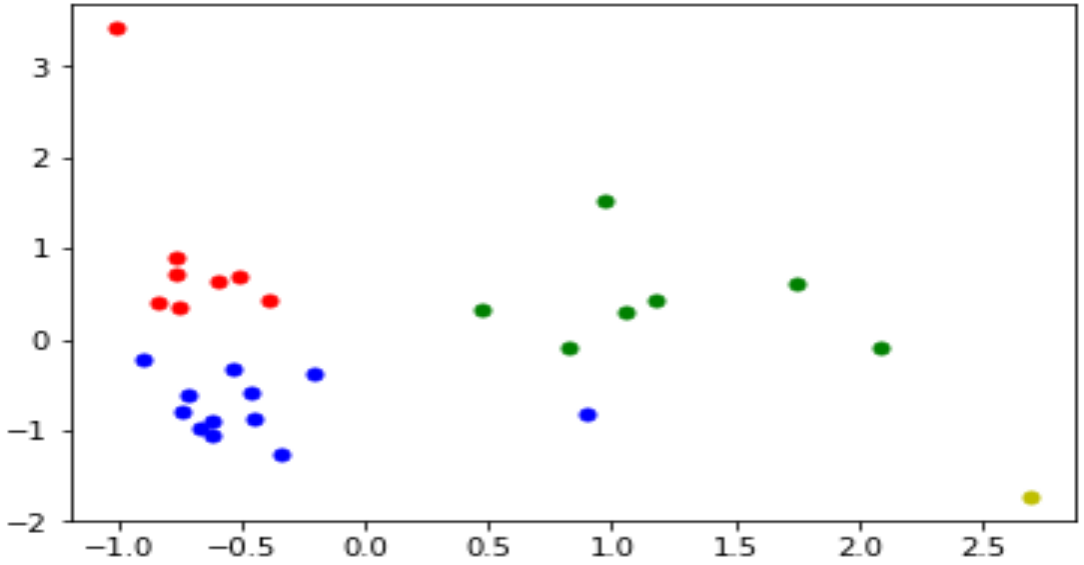
Out[14]:

	Country	Area	GDP	Inflation	Life.expect	Military	Pop.growth	Unemployment	Cluster
0	Austria	83871	41600	3.5	79.91	0.80	0.03	4.2	1
1	Belgium	30528	37800	3.5	79.65	1.30	0.06	7.2	1
2	Bulgaria	110879	13800	4.2	73.84	2.60	-0.80	9.6	2
3	Croatia	56594	18000	2.3	75.99	2.39	-0.09	17.7	2
4	Czech Republic	78867	27100	1.9	77.38	1.15	-0.13	8.5	2

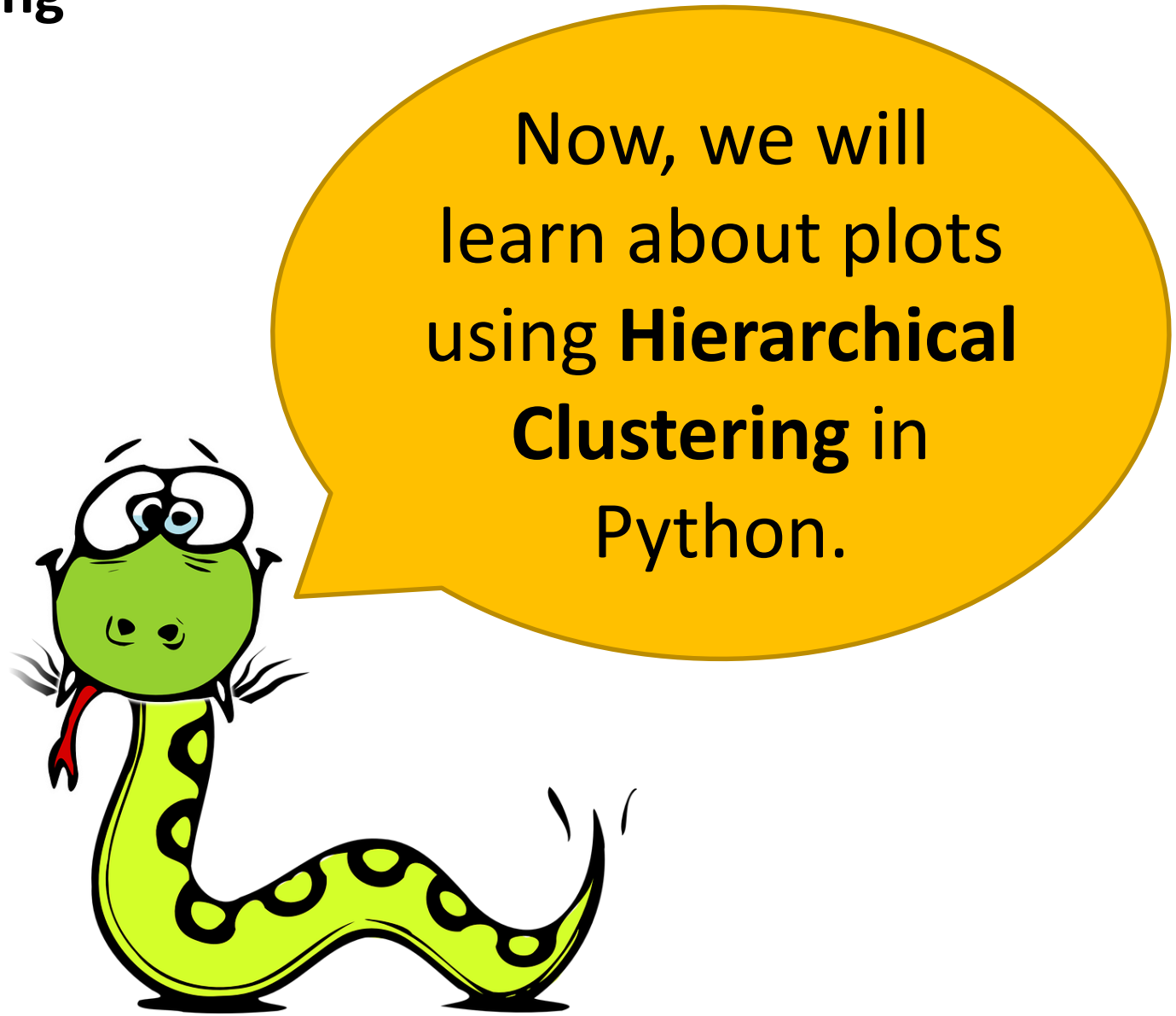
Visualise Cluster

- Visualize the cluster obtained in a 2D plot

In [15]:
```
centers = pd.DataFrame(kmeans.cluster_centers_, columns=dat1.columns)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_
colors = ["g.","r.","b.","y."]

#Plots the clusters in a 2D
X= pd.DataFrame(X_scaled).ix[:,0:1]
for i in range(len(dat1)):
    plt.plot(X.ix[i,0], X.ix[i,1], colors[labels[i]], markersize = 10)
#plt.scatter(centroids[:, 0],centroids[:, 1], marker = "x", s=150, linewidths = 5, zorder = 10)
plt.show()
```



Hierarchical Clustering



Distance Calculation

Calculate the distance using the ‘pdist’ module

```
In [21]: #Hierarchial Clustering
         data_dist = pdist(X_scaled) # computing the distance
         data_dist
```

```
Out[21]: array([ 1.00981728,  4.15989406,  4.2289557 ,  2.05612709,  1.14615801,
                 3.87879176,  2.36524556,  2.14125541,  5.41375608,  2.94603499,
                 1.87952827,  3.19440834,  2.43536697,  3.70513866,  3.31587069,
                 3.59884932,  1.64189127,  2.79505648,  3.32501287,  2.99274871,
                 2.70232784,  2.7093422 ,  4.89473338,  2.6019762 ,  3.07665485,
                 6.52236062,  3.04872653,  3.58224788,  3.36521268,  1.67369445,
                 0.73273801,  3.26321835,  2.10545451,  2.23735292,  4.53251776,
                 2.2944862 ,  2.1863642 ,  2.79862514,  2.14955222,  3.25517726,
                 2.75604062,  3.77282403,  1.46974235,  2.94451197,  2.81466699,
                 2.09449329,  2.06435388,  1.98364423,  4.53441284,  2.68322032,
                 3.21626741,  6.47453766,  2.54487153,  2.79998691,  3.20857023,
```

Linkage

Lets use the ‘complete’ linkage method for combining the records

```
In [22]: data_link = linkage(data_dist, method='complete') # computing the linkage
         data_link
```

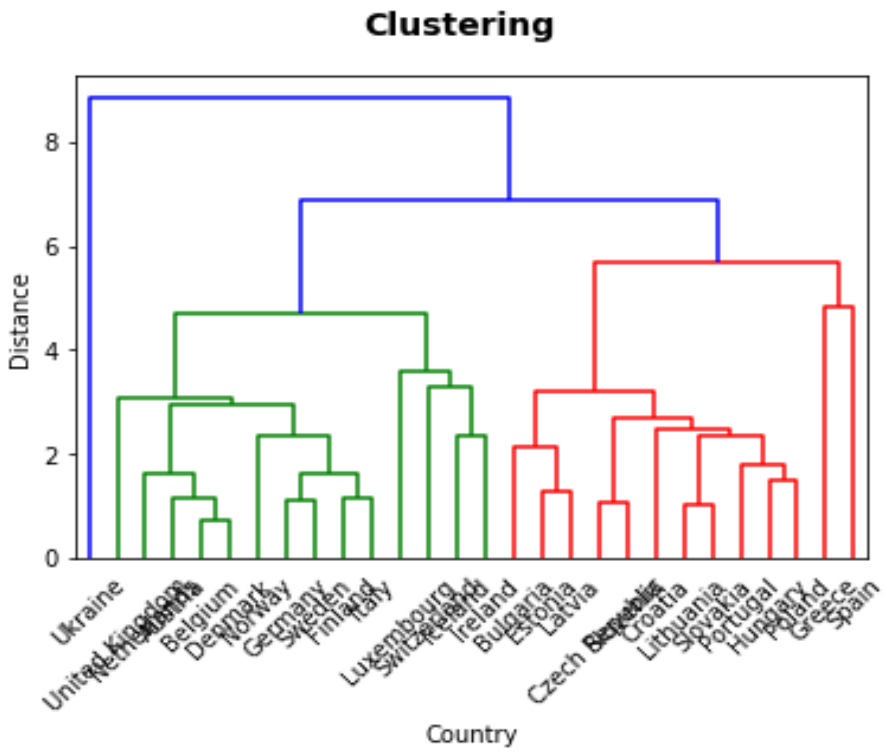
```
Out[22]: array([[  1.        ,   5.        ,   0.73273801,   2.        ],
                [ 15.        ,  21.        ,   1.00881619,   2.        ],
                [  4.        ,  22.        ,   1.08162735,   2.        ],
                [  8.        ,  24.        ,   1.12480261,   2.        ],
                [  0.        ,  28.        ,   1.14615801,   3.        ],
                [  7.        ,  13.        ,   1.17239226,   2.        ],
                [  6.        ,  14.        ,   1.27857317,   2.        ],
                [ 10.        ,  19.        ,   1.48789478,   2.        ],
                [ 31.        ,  33.        ,   1.60443874,   4.        ],
                [ 17.        ,  32.        ,   1.64189127,   4.        ],
                [ 20.        ,  35.        ,   1.78257321,   3.        ],
                [  2.        ,  34.        ,   2.124753  ,   3.        ],
```

Dendogram

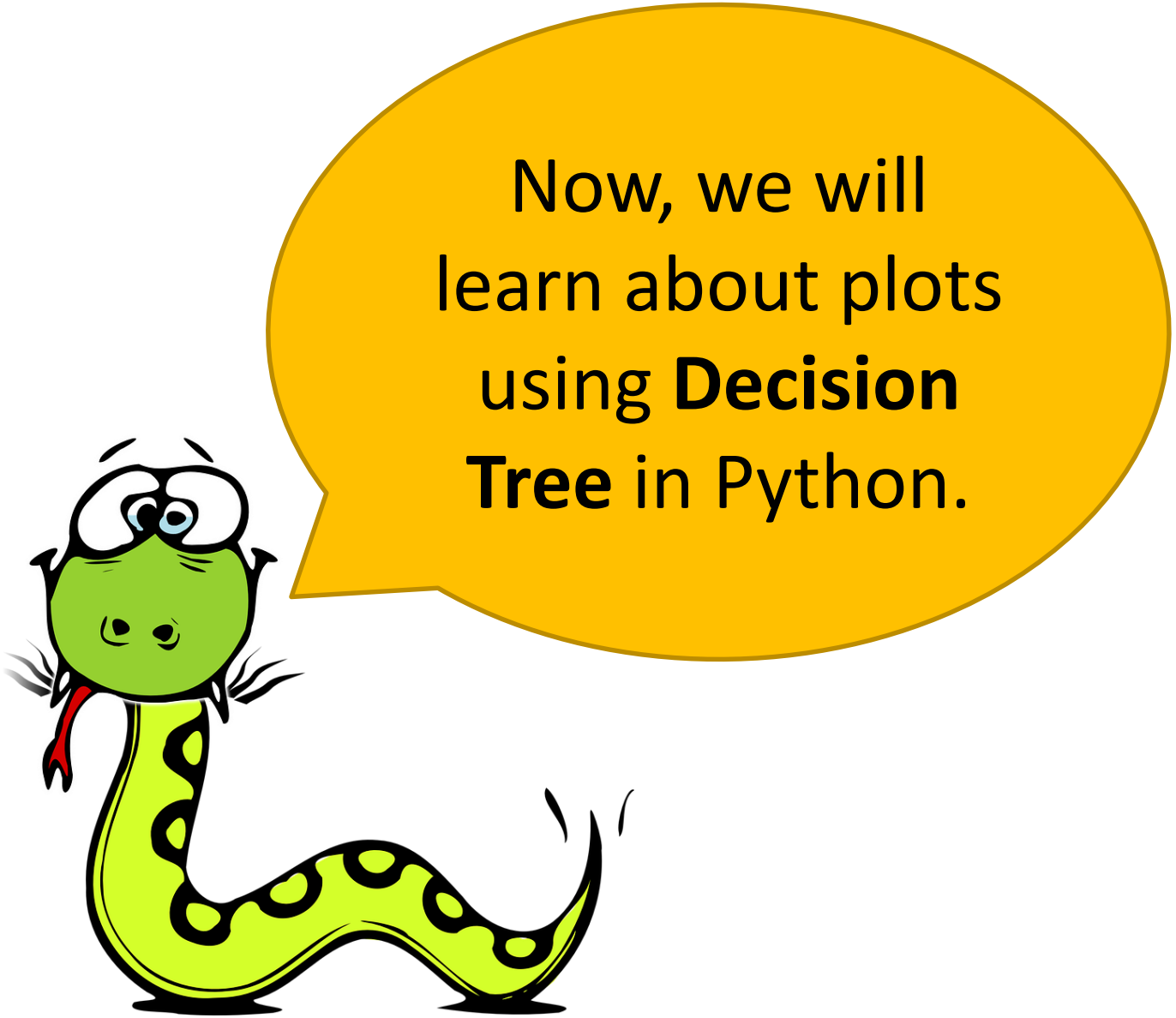
- Plot the dendrogram to determine the number of clusters

```
In [23]: dendrogram(data_link, labels=list(dat.Country))
         plt.xlabel('Country')
         plt.ylabel('Distance')
         plt.suptitle('Clustering', fontweight='bold', fontsize=14)
```

```
Out[23]: Text(0.5,0.98,'Clustering')
```



Decision Tree



Load Packages

- Load the following packages,
  - Numpy – For numeric functions in python
  - Pandas – Package to work with data preparations
  - Sklearn – Scikit learn has several functions for machine learning algorithms
  - tree – Export the tree rules
  - Cross_validation – Module to create train & test split
  - DecisionTreeClassifier – Module to run decision trees
  - GridSearchCV – Module for GridSearch to determine the optimal parameters
  - os – Module for dealing with operating system

```
#importing packages
import pandas as pd
import numpy as py
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import train_test_split
from sklearn import tree
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import os
```

Load Data

- Read the data available in csv file. Data can be loaded from any file types like excel, csv, text, sql etc.

```
In [2]: #Load data
        data = pd.read_csv("E:/IMS/IMS Python/Python/Python/Data Files/binary.csv", header=0)
        data.shape
        data.head()
```

- To see the dimension i.e. the rows & columns of the data use the shape attribute
- To see the top 5 rows of your data use head function,

Out[2]:

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

Data Type Conversion

Convert categorical variables with numeric values to str type. Get all categorical variables and create dummies

In [3]:
```
#Convert categorical variables with numeric values to str type
data['rank'] = data['rank'].astype(str)

#Declare the dependent variable
dep='admit'

#Get all categorical variables and create dummies
obj = data.dtypes == py.object
obj[dep]=False
dummydf = pd.DataFrame()

for i in data.columns[obj]:
    dummy = pd.get_dummies(data[i], drop_first=True)
    dummydf = pd.concat([dummydf, dummy], axis=1)
```

- The “dtypes” gets the data type, using which we pass this column to the “get_dummies” function to convert the categories to dummy columns.
- “drop_first” drops the first category in order to avoid multicollinearity problem
- “pd.concat” combines all the dummy columns for all the categorical variables

Merge Data

- Merge the dummy data with the original data

In [4]:
```
#Merge the dummy and dataset
data1 = data
data1 = pd.concat([data1,dummydf], axis=1)
obj1 = data1.dtypes == py.object
#Create your independent and dependent datasets
X = data1.drop(data1.columns[obj1], axis=1)
X = X.drop([dep], axis=1)

#Appending "V" to columns in order to avoid numeric column names generated while creating dummies
X.columns = 'V_'+X.columns
Y = data1[dep]
```

- “drop” function is used to drop the original column of the categorical variable
- Declare the dependent variable and create your independent and dependent datasets

Train &Test

- Split the data into train & test for model building and validating the model
- We us the model_selection in sklearn package to split the data into train & test
- The “test_size” is used to specify size of the test data, in our case 20%
- “random_state” is used to generate the same set of sample which helps in replicating the results

In [5]:
```
#Split into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)

print('Train Data Size - ', X_train.shape[0], '\n')
print('Test Data Size - ', X_test.shape[0], '\n')
```

```
Train Data Size -  320

Test Data Size -  80
```

Cart

- Since we do not have pruning available in python, we would do a grid search on the parameters to find the best depth where the auc would be maximum
- In general pruning is done on the tree growth, so we can control it by checking for different max_depth
- We check for the performance using the AUC (Area Under Curve)

In [6]:
```
#Run CART algorithm
#Since we do not have pruning available in python, we would do a grid search on the parameters to find the best depth
#where the auc would be maximum
#In general pruning is done on the tree growth, so we can control it by checking for different max_depth
modCART = DecisionTreeClassifier()
param_grid = {'max_depth': py.arange(3, 10)}
gridS = GridSearchCV(modCART, param_grid)
gridS.fit(X_train, Y_train)
tree_preds = gridS.predict_proba(X_test)[:, 1]
tree_performance = roc_auc_score(Y_test, tree_preds)

print('DecisionTree: Area under the ROC curve = {}'.format(tree_performance))
```

```
DecisionTree: Area under the ROC curve = 0.5445156695156695
```

Grid Search Results

The best depth which gives maximum auc can be obtained using the 'best_params_' attribute

```
In [7]: #The best depth which gives maximum auc
        gridS.best_params_

        #View all the scores for different depth combinations
        gridS.grid_scores_
```

To view all the scores for different depth combinations use the 'grid_scores_' attribute

```
Out[7]: [mean: 0.67500, std: 0.01497, params: {'max_depth': 3},
         mean: 0.66875, std: 0.01063, params: {'max_depth': 4},
         mean: 0.66250, std: 0.00778, params: {'max_depth': 5},
         mean: 0.68125, std: 0.01893, params: {'max_depth': 6},
         mean: 0.64375, std: 0.02877, params: {'max_depth': 7},
         mean: 0.62813, std: 0.03595, params: {'max_depth': 8},
         mean: 0.63438, std: 0.02134, params: {'max_depth': 9}]
```

Finalize The Tree

We finalise the model on max_depth of 3

In [8]:
```
#We finalise the model on max_depth of 6
modCART = DecisionTreeClassifier(max_depth=6)
modCART.fit(X_train, Y_train)
```

Out[8]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```

Method to install graphviz package

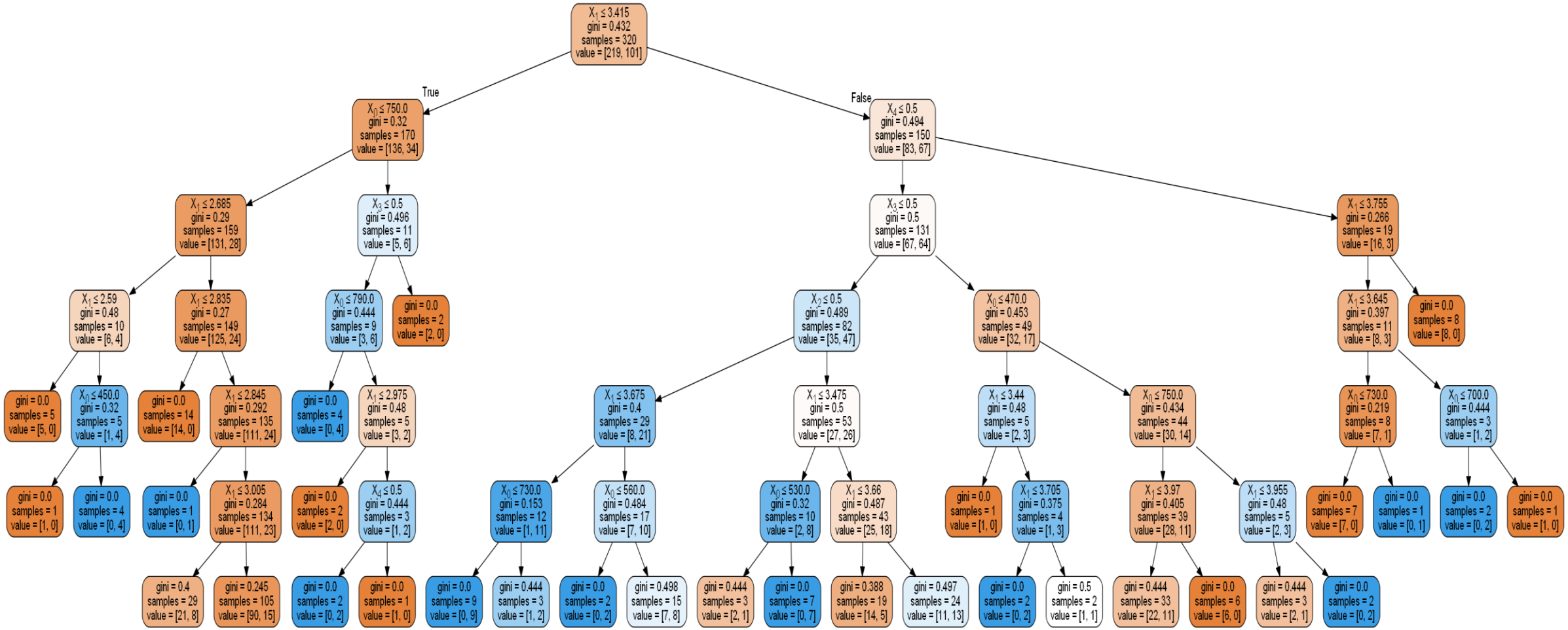
1. Download "graphviz-2.38.msi" from http://www.graphviz.org/Download_windows.php
2. Execute the "graphviz-2.38.msi" file
3. Add the graphviz bin folder to the PATH system environment variable (Example: "C:\Graphviz2.38\bin")
4. Go to Anaconda Prompt using start menu (Make sure to right click and select "Run as Administrator". We may get permission issues if Prompt as not opened as Administrator)
5. Execute the command: conda install graphviz
6. Pip install graphviz in the command line
7. Execute the command "conda list" and make sure pydot and graphviz modules are listed. Thanks

Visualize The Tree

- Visualize the tree
- Export the rules to '.dot' data and convert it to png

In [17]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
%matplotlib inline
import pydotplus
import pydot
from sklearn.tree import export_graphviz
dot_data = StringIO()
export_graphviz(modCART, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Decision Tree



Thank You.