PYTHON FOR DATA MINING

TOOL

- Jupyter Notebook
  - *https://www.dataquest.io/blog/jupyter-notebook-tutorial/*
  - *https://realpython.com/jupyter-notebook-introduction/*
- Colab
- Pycharm

PYTHON BASIC

PYTHON TUTORIAL

- Programming assignments will be in Python. If you are not familiar with Python, you are expected to bring yourself up to the mark!

- A good place to start is python.org. You can start with their Python Tutorial.
  - *There are many others to choose from.*

- Most Python tutorials expect you to write and run Python code as you go along,
  - *This kind of active learning is the way to go.*

PYTHON BASICS

- **Python basic**
  - *https://www.python.org/*

- **Python Tutorials**
  - *https://www.tutorialspoint.com/python/index.htm*
  - *https://www.w3schools.com/python/python_numbers.asp*

- The Python Package Index (PyPI) is a repository of software for the Python programming language.

PYTHON BASICS

Python developed in the late eighties is a great language for the beginner-level programmers

Python is Interpreted

- *Python is processed at runtime by the interpreter. No need to compile your program before executing it.*

Python is Interactive

- *You can interact with the interpreter directly to write your programs.*

Python is Object-Oriented

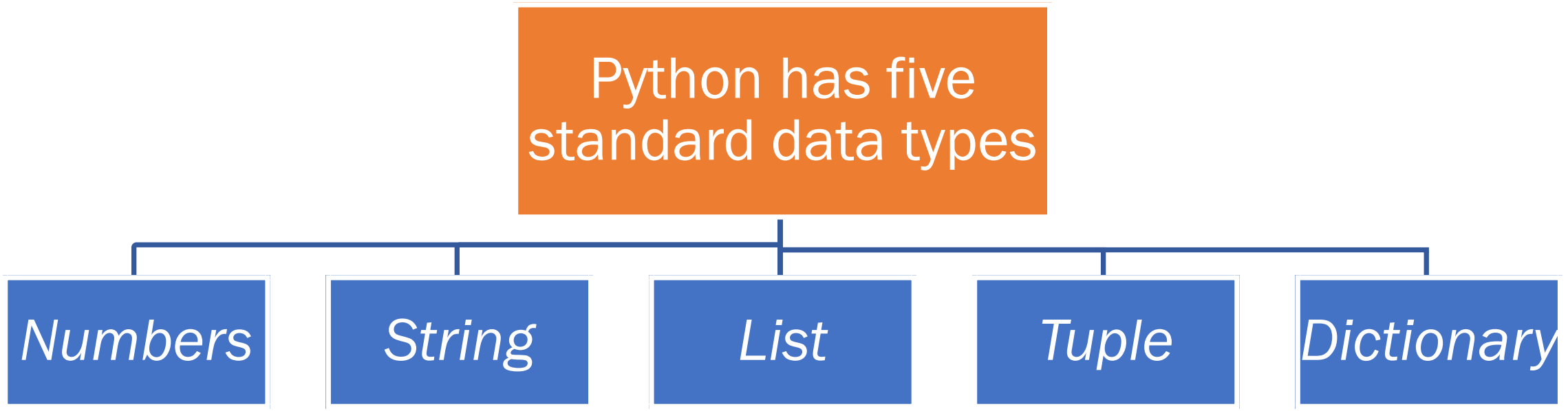
PYTHON BASICS

- **Python Indentations**
  - *Python uses indentation to indicate a block of code.*

- **Python Variables**
  - *Variables do not need to be declared with any particular type and can even change type after they have been set.*

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

PYTHON CORE DATA TYPES



PYTHON NUMBER AND STRING

- Numbers: three numeric types in Python:
  - *int*
  - *float*
  - *complex*

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
print(type(x)) # tell type of x
```

- String

```
str = 'Hello World!'

print(str)          # Prints complete string
print(str[0])       # Prints first character of the string
print(str[2:5])     # Prints characters from 3rd to 5th
print(str[2:])      # Prints string from 3rd character
print(str * 2)      # Prints string two times
print(str + "TEST") # Prints concatenated string
```

```
OUTPUT

Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

PYTHON LIST

- List
  - *Lists are similar to arrays in C++ (to some extent).*
  - *One difference between them is that all the items belonging to a list can be of different data type.*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements from 2nd till 3rd
print list[2:]      # Prints elements from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

```
Output


['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123,'john']
```

PYTHON TUPLE

- TUPLE
  - *A tuple consists of a number of values separated by commas.*
  - *It is similar to the list but cannot be updated.*
    - Tuples can be thought of as **read-only** lists.
  - *Unlike lists, tuples are enclosed within parentheses.*

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple               # Prints complete list
print tuple[0]            # Prints first element of the list
print tuple[1:3]          # Prints elements from 2nd till 3rd
print tuple[2:]           # Prints elements from 3rd element
print tinytuple * 2       # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

OUTPUT

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

PYTHON **DICTIONARY**

- Dictionary
  - *Python's dictionaries are kind of hash table type.*
  - *They work like associative arrays consisting of [key, value] pair*

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict={'name':'john','code':6734,'dept':'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

```
OUTPUT



This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

PYTHON IF

```
var = 100
if ( var == 100 ) :
    print( "Value of expression is 100")
    print ("Good bye!")
else :
    print("else part")
```

- Unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.
- Core Python does not provide switch or case statements

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

PYTHON WHILE LOOP

```
count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1
```

- **Using else Statement with Loops**
  - *If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.*

```
count = 0
while (count < 5):
   print ('The count is:', count)
   count = count + 1
else :
   print (count, " is not less than 5")
```

```
i = 2
while(i < 100):
   j = 2
   while(j <= (i/j)):
      if not(i%j): break
      j = j + 1
   if (j > i/j) : print (i, " is prime")
   i = i + 1
```

PYTHON FOR LOOP

```
for x in 'Python':     # First Example
   print ('Current Letter :', x)

fruits = ['banana', 'apple',  'mango']
for x in fruits:         # Second Example
   print ('Current fruit :', x)
```

for *target* in *object*:	*# Assign object items to target*
statements	*# Repeated loop body: use target*
else:	*# Optional else part*
statements	*# If we didn't hit a 'break*

```
for num in range(0,10):     #to iterate between 0 to 10
   for i in range(2,num):    #to iterate on the factors of the number
      if num%i == 0:         #to determine the first factor
         j=num/i             #to calculate the second factor
         print ('%d equals %d * %d' % (num,i,j))
         break #to move to the next number, the #first FOR
   else:                   # else part of the loop
      print (num, 'is a prime number')
```

https://www.tutorialspoint.com/python/python_loops.htm

PYTHON FUNCTIONS

- Creating a Function

```
def my_function():
    print("Hello from a function")
```

- Calling a Function

```
my_function()
```

- Parameters

```
def my_function(fname):
    print(fname + " the name")
```

- Returning a Value from a Function

```
def my_function(x):
    return 5 * x
```

https://www.w3schools.com/python/python_functions.asp

PYTHON LAMBDA FUNCTIONS

- **Python Lambda**
  - *A lambda function is a small anonymous function.*
  - *A lambda function can take any number of arguments but can only have one expression.*

A lambda function that multiplies argument a with argument b and print the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

A lambda function that sums argument a, b, and c and print the result:

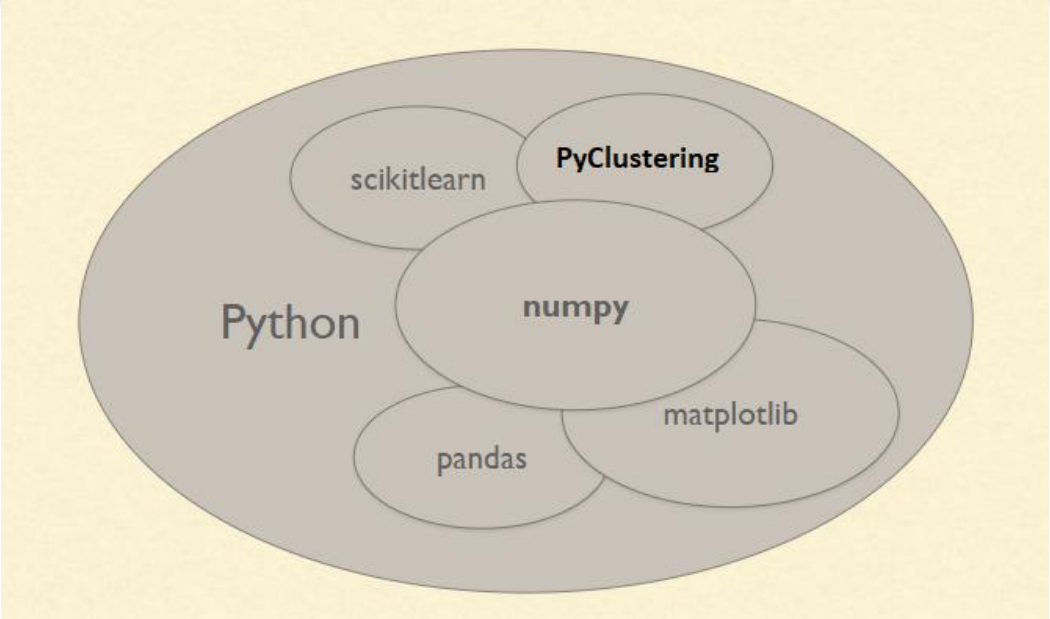
```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

https://www.w3schools.com/python/python_functions.asp

DATA MINING AND ANALYSIS

DATA ANALYSIS

- Consist of
  - *Data Preprocessing*
  - *Clustering*
  - *Visualization*



Numpy

- Provides fast mathematical computation on arrays and matrices.

Pandas

- Provides high-performance, easy to use structures and data analysis tools.
- Pandas provides in-memory 2d table object called Dataframe.

Matplotlib

- Matplotlib is a 2d plotting library for producing quality figures

NUMPY

- NumPy stands for ‘Numerical Python’ or ‘Numeric Python’.
- Numpy is the core library for scientific computing in Python.
- It provides a high-performance multidimensional array object, and tools for working with these arrays

http://cs231n.github.io/python-numpy-tutorial/

https://www.tutorialspoint.com/numpy

https://cloudxlab.com/blog/numpy-pandas-introduction/

NUMPY ARRAYS

- A numpy array is a grid of values all of the same type
- **RANK**
  - *The number of dimensions is the rank of the array*
- **SHAPE**
  - *It is a tuple of integers giving the size of the array along each dimension.*

```
import numpy as np
a = np.array([1, 2, 3])   # Create a rank 1 array
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

CREATE NUMPY ARRAY

- Numpy also provides many functions to create arrays:

```
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)
```

http://cs231n.github.io/python-numpy-tutorial/

NUMPY ARRAY INDEXING

- NumPy offers several ways to index into arrays.
- **Slicing:** Similar to Python lists, numpy arrays can be sliced.
  - *Since arrays may be multidimensional, you must specify a slice for each dimension of the array:*

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

A slice of an array is a view into the same data, so modifying it will modify the original array.

```
b = a[1:]
```

```
B is a subarray consisting of the row 1 and onwards
b is the array of shape (2, 4)
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

NUMPY ARRAY INDEXING

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

```
c = a[..., 1]
```

```
C is a subarray consisting of the second column of a
#  [2 6 10]
```

```
d = a[1, ...]
```

```
d is a subarray consisting of the second row of a
#  [5 6 7 8]
```

```
f = a[..., 1:]
```

```
f is a subarray consisting of the column 1 onwards values in a
# [[ 2  3  4]
#  [ 6  7  8]
#  [ 10 11 12]]
```

NUMPY ARRAY INDEXING

- NumPy offers several ways to index into arrays.
- **Slicing:** Similar to Python lists, numpy arrays can be sliced.

A slice of an array is a view into the same data, so modifying it will modify the original array.

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

```
c = a[1:3, 1:3]
```

```
C is a subarray consisting of the row from 1 & 2 =>[1:3] means rows before 3
and columns 1 and 2[1:3] means cols start from and ends before 3
c is the array of shape (2, 2)
# [[6 7]
#  [10 11]]
```

NUMPY ARRAY MATH

- Basic mathematical functions operate elementwise on arrays and are available both as operator overloads and as functions in the numpy module.

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

print(x + y)
print(x - y)
print(x * y) # Elementwise product, not matrix multiplication
print(x / y)
print(np.sqrt(x))# Elementwise square root
Print(np.dot(x,y)) # Matrix multiplication
```

```
print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]“

print(x.T)  # Compute transpose
```

Numpy provides many more functions for manipulating arrays; you can see the full list in the documentation.

NUMPY BROADCAST

- Broadcasting is a powerful mechanism that allows Numpy to work with arrays of different shapes when performing arithmetic operations.
- For example, suppose that we want to add a constant vector to each row of a matrix.

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v
```

SciPy

Numpy provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays.

SciPy builds on this and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications.

NUMPY SCIPY

- SciPy builds on NumPy and provides a large number of functions that operate on numpy arrays
- It is useful for different types of scientific and engineering applications.
- Image Processing
  - *SciPy provides some basic functions to work with images. For example, it has functions*
    - to read images from disk into numpy arrays,
    - to write numpy arrays to disk as images,
    - to resize images.
- **Distance between points**
  - *SciPy defines some useful functions for computing distances between sets of points.*

NUMPY SCIPY

- **Distance between points**
  - *SciPy defines some useful functions for computing distances between sets of points.*

```
import numpy as np
from scipy.spatial.distance import pdist, squareform

x = np.array([[0, 1], [1, 0], [2, 0]])
d = squareform(pdist(x, 'euclidean'))
# Compute the Euclidean distance between all rows of x.
# d[i, j] is the Euclidean distance between x[i, :] and x[j, :],
# and d is the following array:
# [[ 0.          1.41421356  2.23606798]
#  [ 1.41421356  0.          1.        ]
#  [ 2.23606798  1.          0.        ]]
```

A similar function (scipy.spatial.distance.cdist) computes the distance between all pairs across two sets of points; you can read about it in the documentation.

PANDAS

https://www.tutorialspoint.com/python_pandas/python_pandas_introduction.htm

PANDA

- The name Pandas is derived from the word **Pa**nel **Da**ta – an Econometrics from Multidimensional data.
- It is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- **Key Features of Pandas**
  - *Fast and efficient DataFrame object*
  - *Tools for loading data into in-memory data objects from different file formats.*
  - *Data alignment and integrated handling of missing data.*
  - *Group by data for aggregation and transformations.*
  - *High performance merging and joining of data.*
  - *Time Series functionality.*

PANDA

- Standard Python distribution doesn't come bundled with Pandas module.
- A lightweight alternative is to install NumPy using popular Python package installer, pip.
  - *pip install pandas*
- If you install Anaconda Python package, Pandas will be installed by default
  - ***Anaconda** is a free Python distribution for SciPy stack.*

PANDA DATA-STRUCTURES

- Pandas deals with the following three data structures –
  - *Series*
  - *DataFrame*
  - *Panel*

- These data structures are fast as they are built on top of NumPy array.

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, sizeimmutable.
Data Frames	2	2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	3D labeled, heterogeneous and size-mutable array.

PANDA DATA-FRAME

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)

data1 = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data1,columns=['Name','Age'])
print df

for col in df: # iteration in df
   print col
```

DataFrame can be created using various inputs

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

PANDA DATA-FRAME

- Functionality
  - Descriptive statistics
    - Methods to compute sum, mean, std etc
  - Function Application
    - Can apply functions to Pandas objects
  - Reindexing
  - Sorting
  - Iteration
  - Indexing (slice) & selecting Data
  - GroupBy
  - Joining Data-Frames
  - Helps to handle missing data (check, drop, replace, statistics)

PYTHON AND DATA PRE-PROCESSING

http://scikit-learn.org/stable/modules/preprocessing.html

DATA PREPROCESSING

- **sklearn.preprocessing** package provides several common utility functions for Data Preprocessing
  - *Standardization*
  - *Normalization*

```
from sklearn import preprocessing
import numpy as np
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
X_scaled = preprocessing.scale(X_train)

X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Scaling features to a range

```
min_max_scaler = preprocessing.MinMaxScaler()
```

Scale to [0 , 1] range. You can specify the min and max value too

DATA PREPROCESSING

- **Discretization:** provides a way to partition continuous features into discrete values.
  - *K-bins discretization*
    - discretizers features into k equal width bins:

```
X = np.array([[ -3., 5., 15 ],
              [  0., 6., 14 ],
              [  6., 3., 11 ]])

est = preprocessing.KBinsDiscretizer(n_bins=[3, 2, 2], encode='ordinal').fit(X)
```

DATA PREPROCESSING

- **Encoding categorical features**
  - *Such features can be efficiently coded as integers*
- For example:
  - *A person could have features*
    - ["male", "female"],
    - ["from Europe", "from US", "from Asia"],
    - ["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]
  - ***Encoding***
    - ["male", "from US", "uses Internet Explorer"] could be expressed as [0, 1, 3] while
    - ["female", "from Asia", "uses Chrome"] would be [1, 2, 1].

PYTHON AND CLUSTERING

BASIC TERMS AND RESOURCES

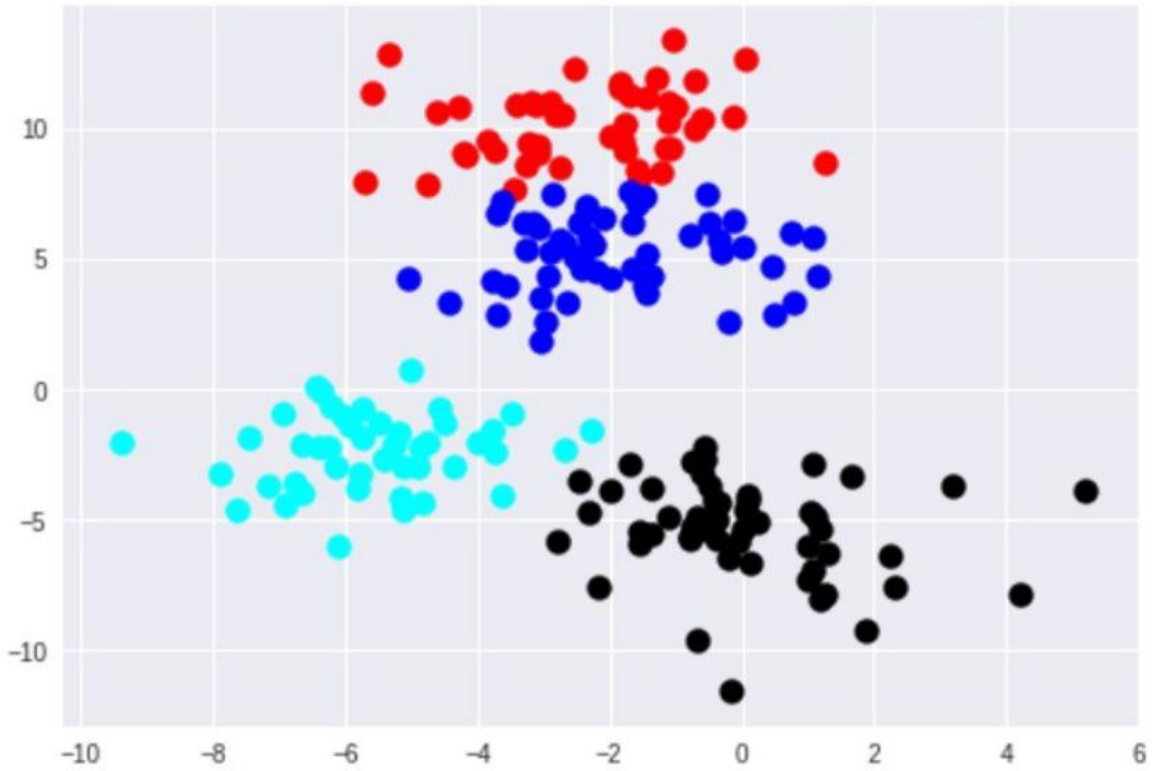
- PyClustering
  - *It is a Python and C++ data mining library*
  - *https://pypi.org/project/pyclustering/*
- Sklearn.cluster
  - *This module gathers popular unsupervised clustering algorithms.*
  - *http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster*
- Very Useful tutorial
  - *https://towardsdatascience.com/an-introduction-to-clustering-algorithms-in-python-123438574097*

K-MEANS

```
# import KMeans
from sklearn.cluster import KMeans
# create kmeans object
kmeans = KMeans(n_clusters=4)

# fit kmeans object to data
kmeans.fit(points)

# print location of clusters learned by kmeans object
print(kmeans.cluster_centers_)

# save new clusters for chart
y_km = kmeans.fit_predict(points)
```

K MEANS

```
plt.scatter(points[y_km ==0,0], points[y_km == 0,1], s=100, c='red')
plt.scatter(points[y_km ==1,0], points[y_km == 1,1], s=100, c='black')
plt.scatter(points[y_km ==2,0], points[y_km == 2,1], s=100, c='blue')
plt.scatter(points[y_km ==3,0], points[y_km == 3,1], s=100, c='cyan')
```



BASIC CLUSTERING ALGORITHMS - PYCLUSTERING

PyClustering

It is a Python and C++ data mining library
https://pypi.org/project/pyclustering/

Sklearn.cluster

This module gathers popular unsupervised clustering algorithms.
http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster

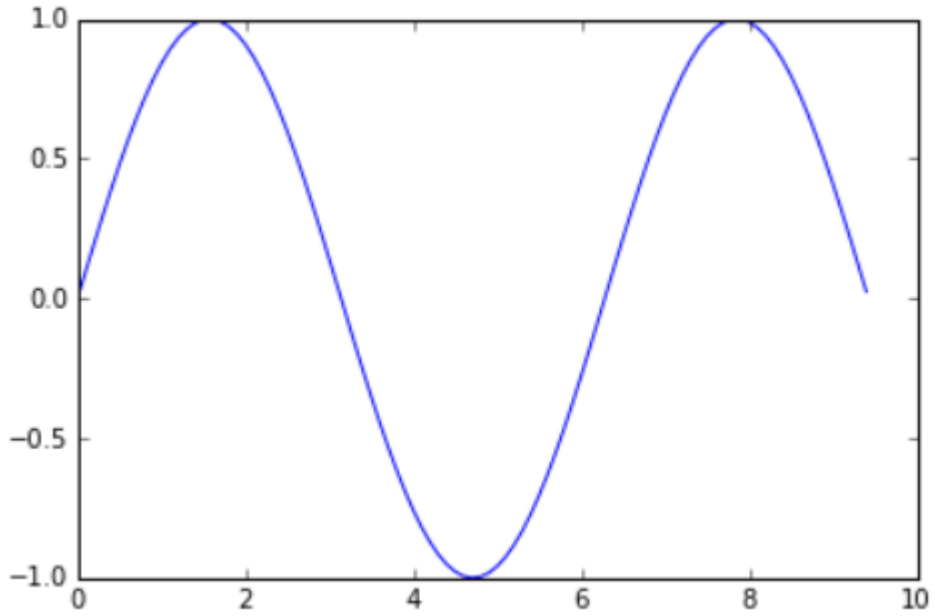
- **Agglomerative** (pyclustering.cluster.agglomerative);
- **BIRCH** (pyclustering.cluster.birch);
- **CLARANS** (pyclustering.cluster.clarans);
- **CURE** (pyclustering.cluster.cure);
- **DBSCAN** (pyclustering.cluster.dbscan);
- **GA (Genetic Algorithm)** (pyclustering.cluster.ga);
- **K-Means** (pyclustering.cluster.kmeans);
- **K-Means++** (pyclustering.cluster.center_initializer);
- **K-Medians** (pyclustering.cluster.kmedians);
- **K-Medoids (PAM)** (pyclustering.cluster.kmedoids);
- **OPTICS** (pyclustering.cluster.optics);
- **ROCK** (pyclustering.cluster.rock);
- **SOM-SC** (pyclustering.cluster.somsc);

PYTHON AND DATA VISUALIZATION

MATPLOTLIB

- Matplotlib is a plotting library.
- The matplotlib.pyplot module provides a plotting system similar to that of MATLAB.

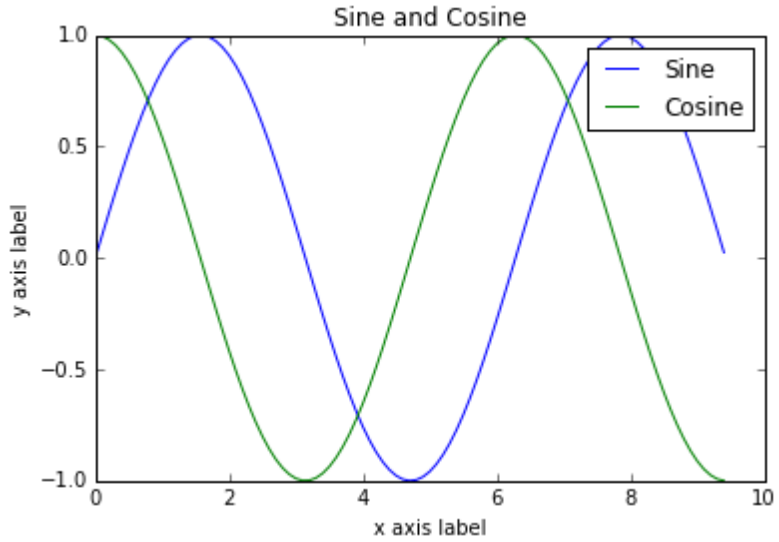
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for
points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to
make graphics appear.
```



MATPLOTLIB

We can easily plot multiple lines at once, and add a title, legend, and axis labels

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine
and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



MATPLOTLIB - SUBPLOTS

```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid with height=2 width=1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```